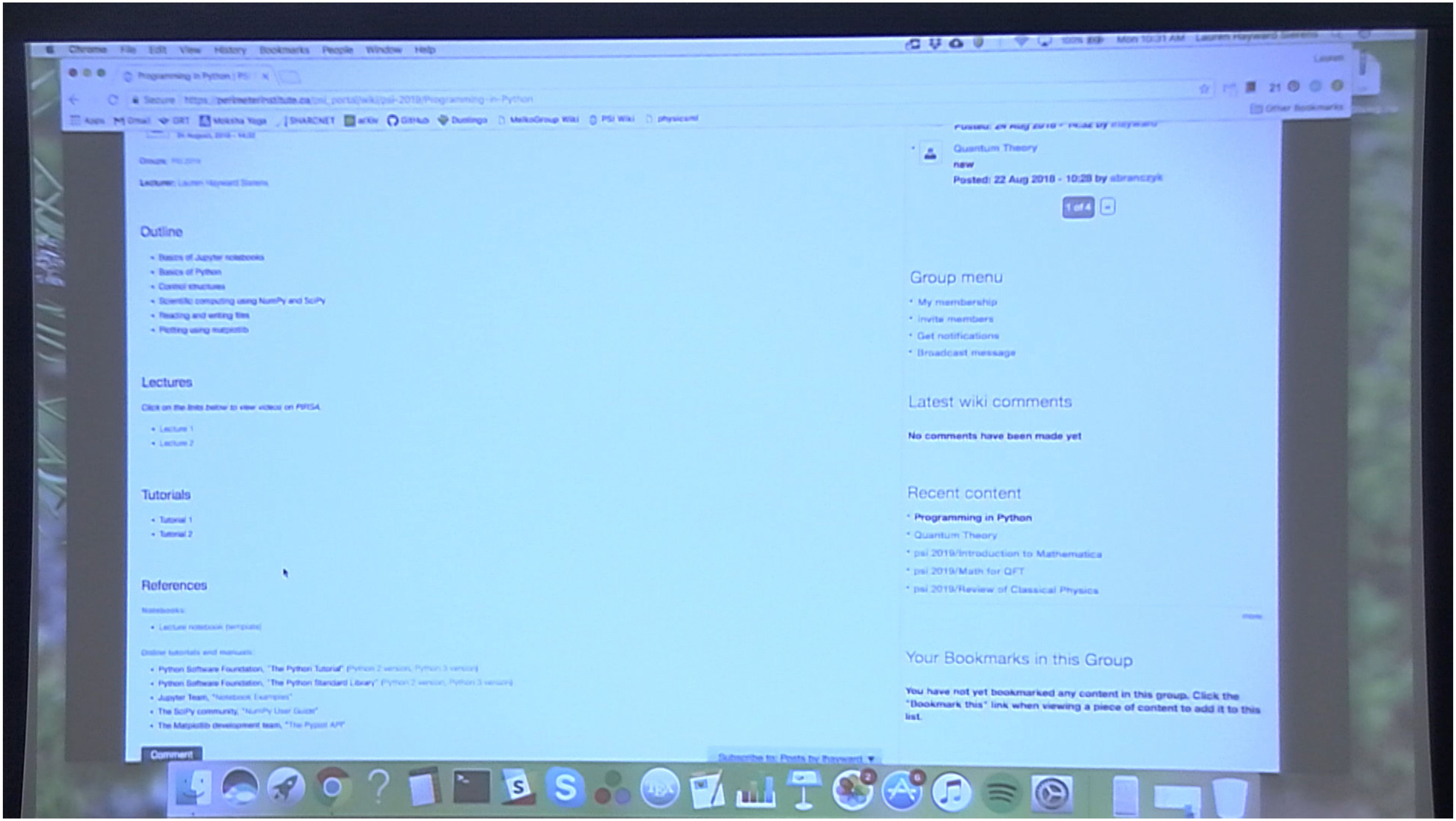
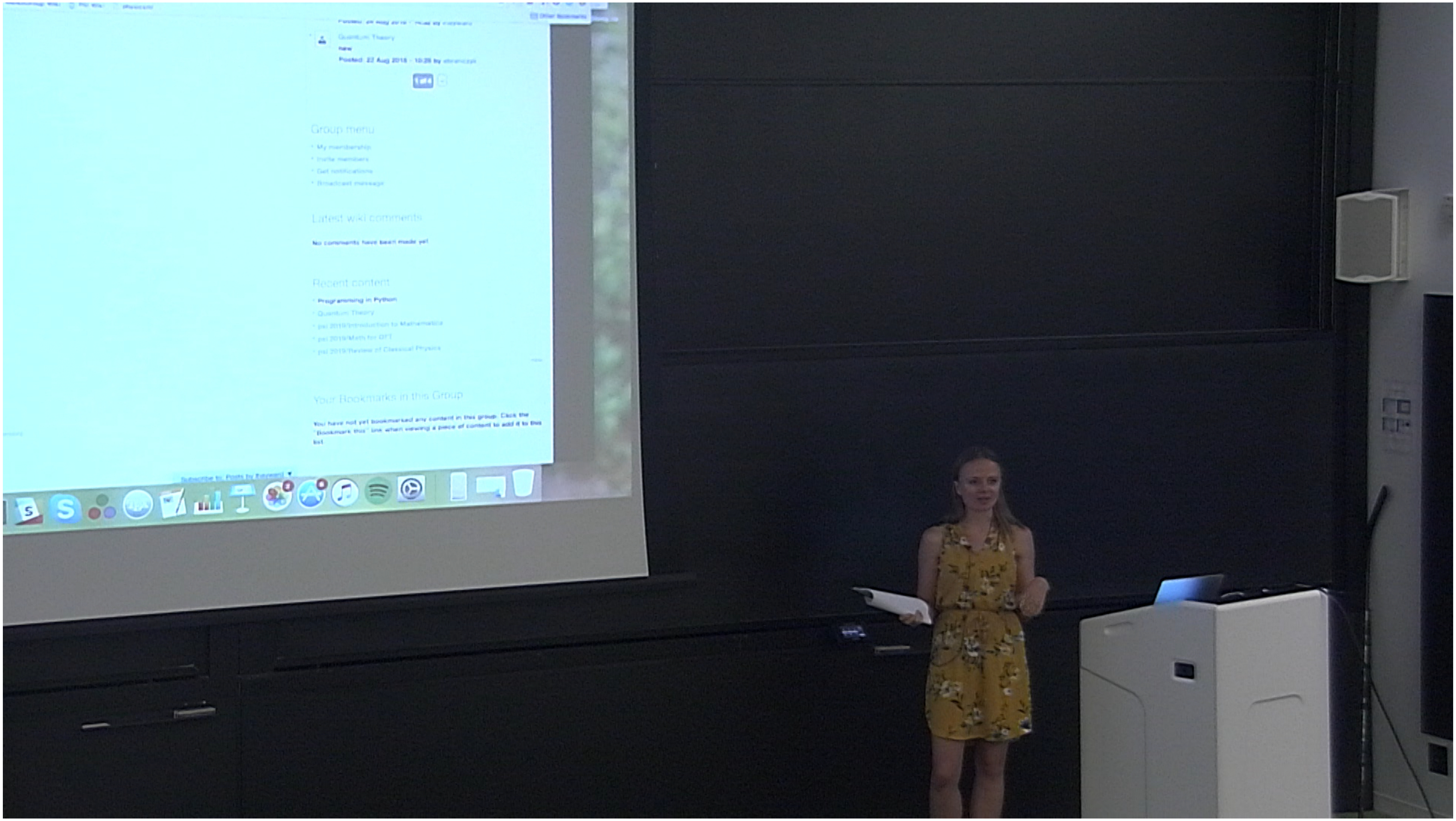
Title: PSI 2018/2019 - Programming in Phyton - Lecture 2

Date: Aug 27, 2018 10:30 AM

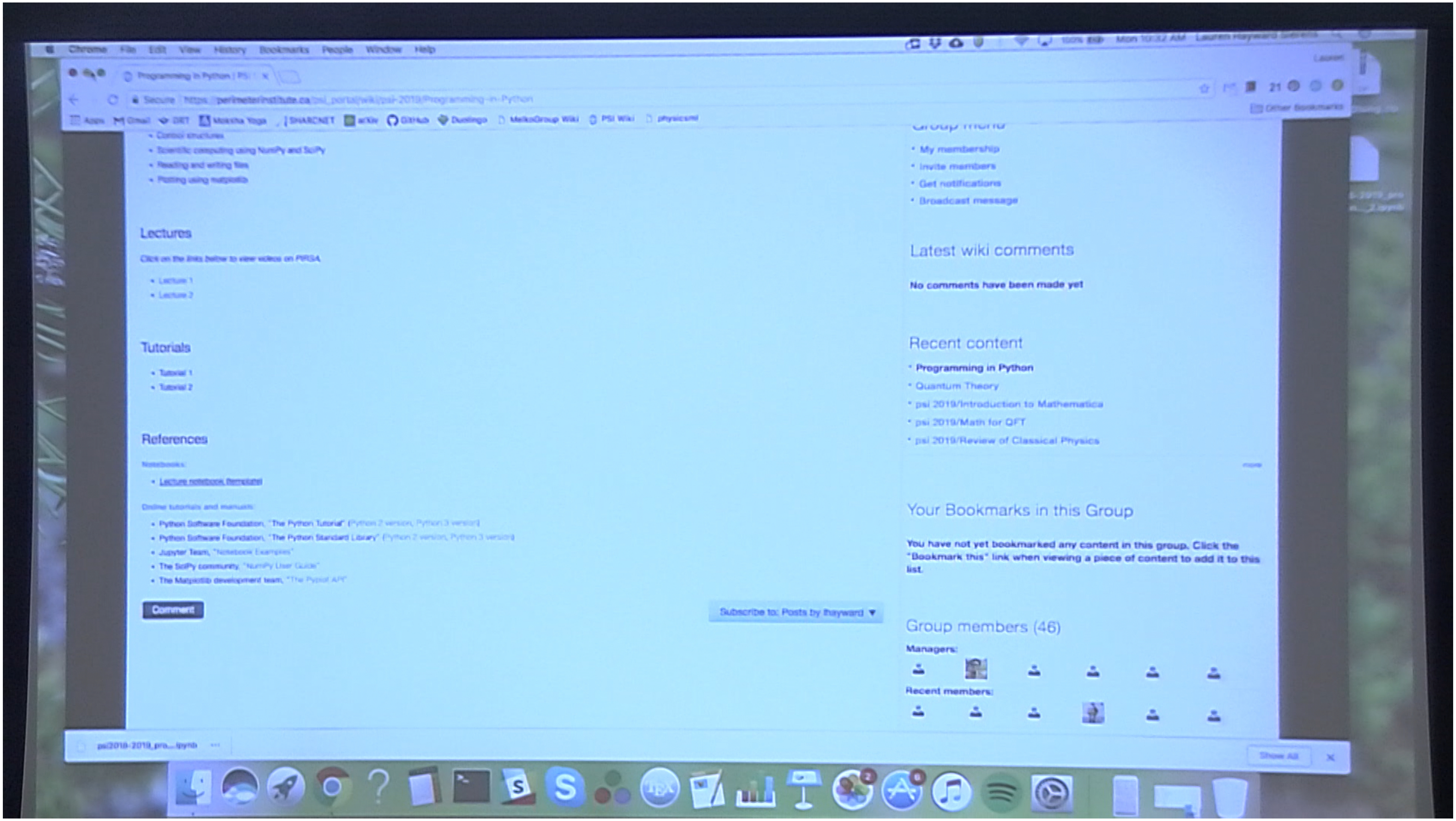
URL: http://pirsa.org/18080024

Abstract:

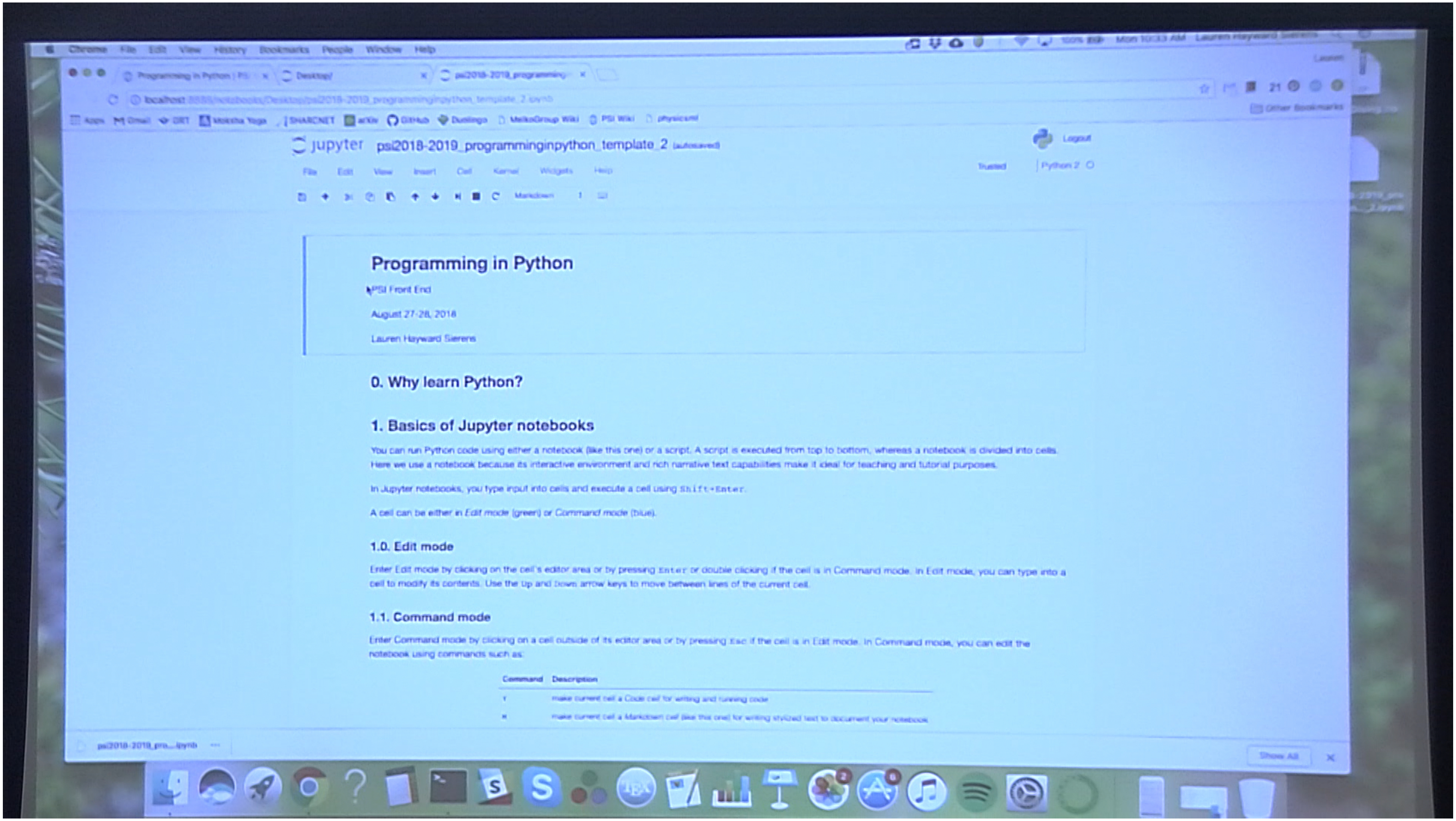

Programming in Python
Outline
Basics of Jupyter notebooks
Basics of Python
Control structures
Scientific computing using NumPy and SciPy
Reading and writing files
Plotting using matplotlib
Lectures
Click on the links below to view videos on PIRSA.
Lecture 1
Lecture 2
Tutorials
Tutorial 1
Tutorial 2
References
Python Software Foundation, "The Python Tutorial"
Python Software Foundation, "The Python Standard Library"
Jupyter Team,
The SciPy community,
The Matplotlib development team,
Comment
Quantum Theory
new
Posted: 22 Aug 2018 - 10:28 by abranczyk
Group menu
My membership
Invite members
Get notifications
Broadcast message
Latest wiki comments
No comments have been made yet
Recent content
Programming in Python
Quantum Theory
psi 2019/Introduction to Mathematica
psi 2019/Math for QFT
psi 2019/Review of Classical Physics
Your Bookmarks in this Group
You have not yet bookmarked any content in this group. Click the "Bookmark this" link when viewing a piece of content to add it to this list.

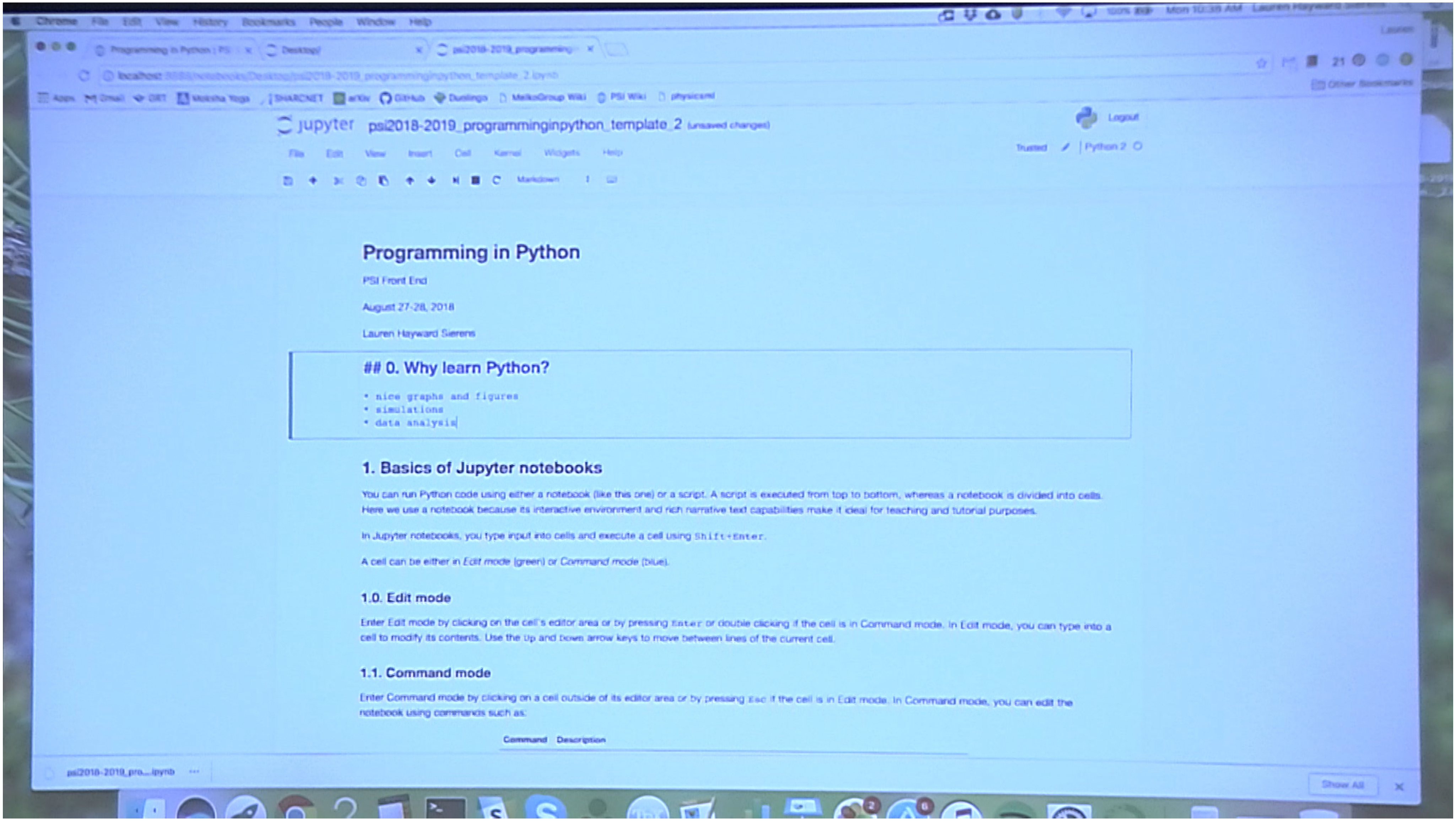


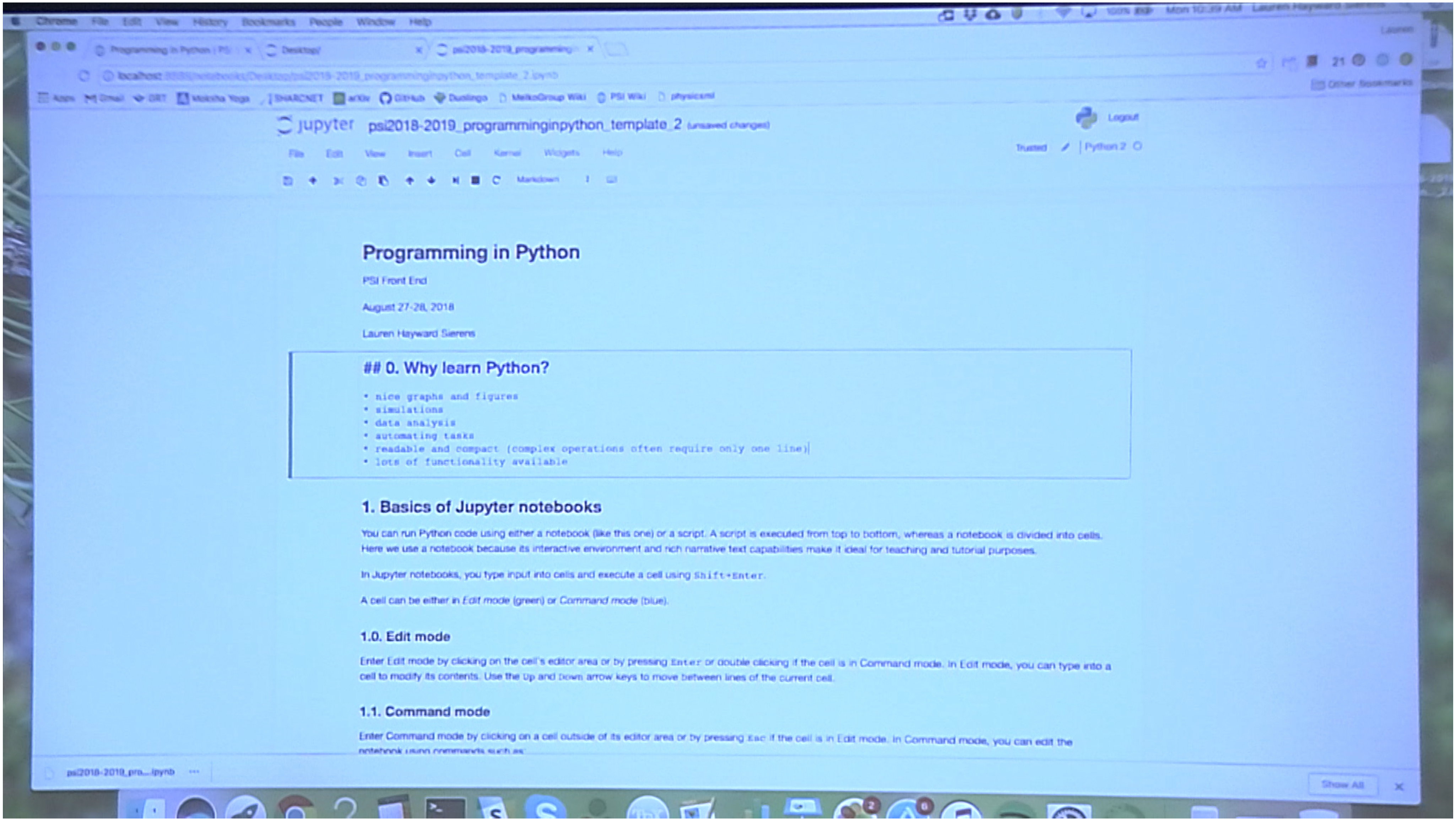
Group menu
Latest wiki comments
No comments have been made yet
Recent content
Programming in Python
Quantum Theory
Your Bookmarks in this Group

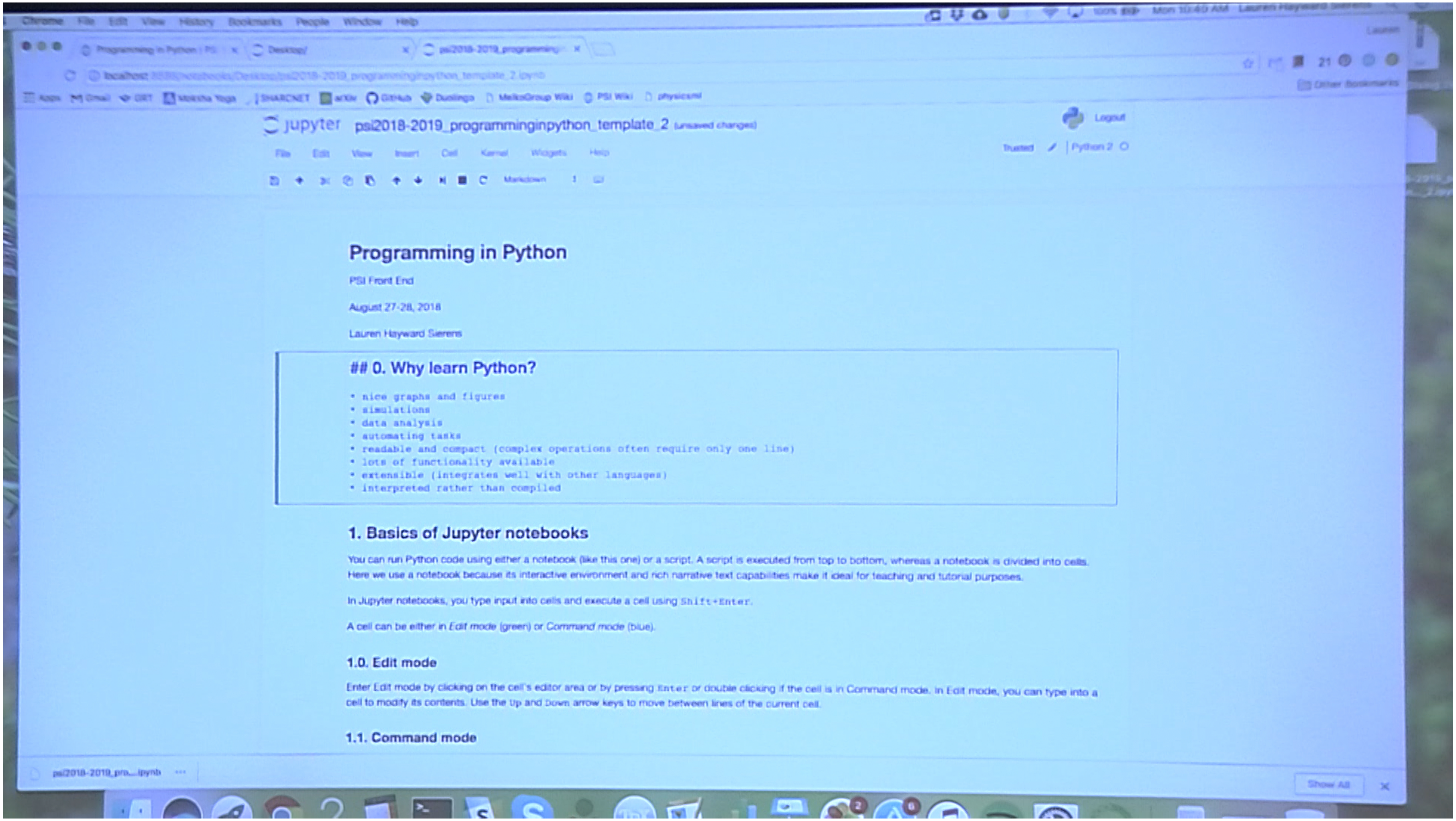


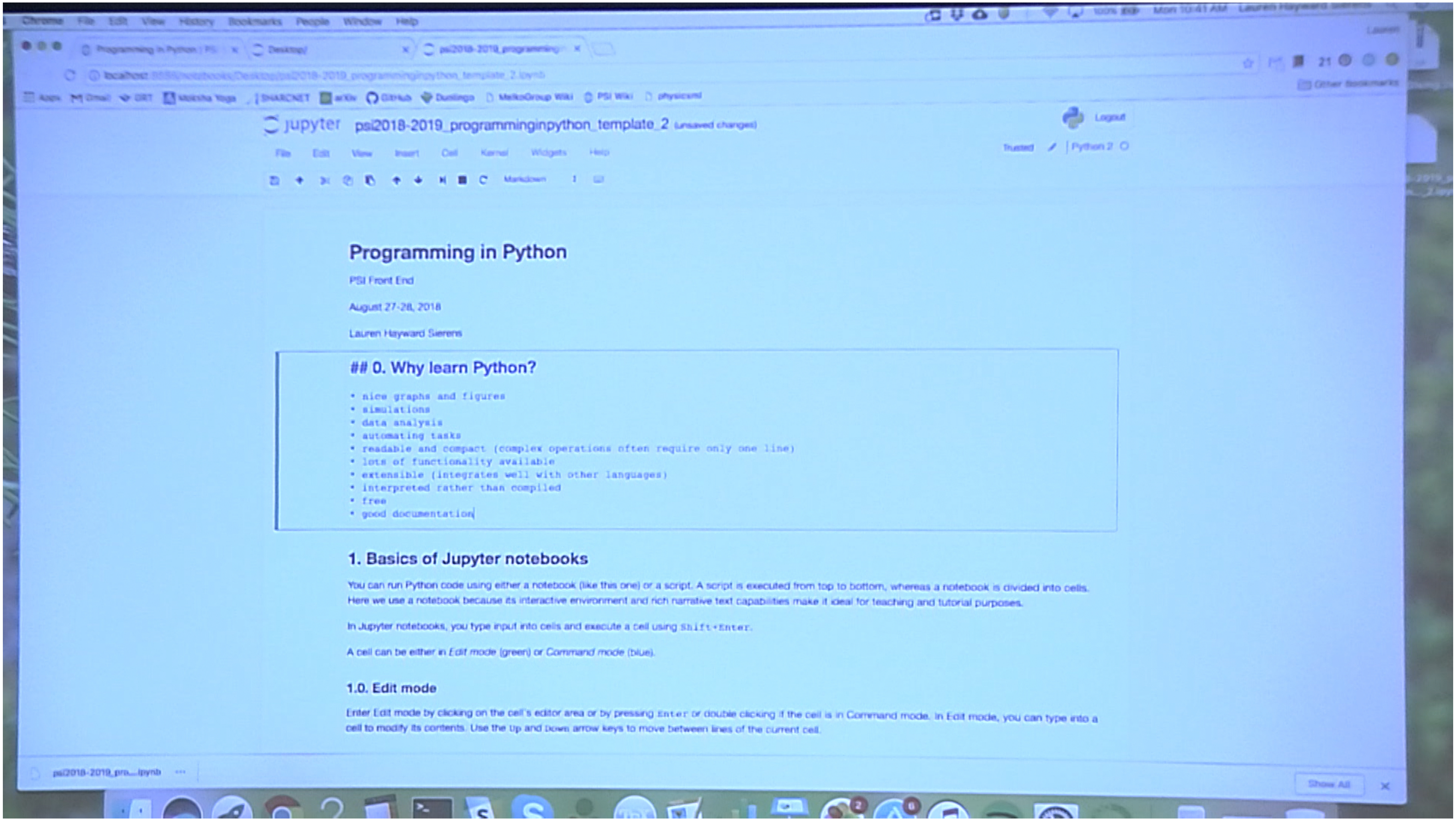
Programming in Python | PSI
Control structures
Scientific computing using NumPy and SciPy
Reading and writing files
Plotting using matplotlib
Lectures
Click on the links below to view videos on PIRSA
Lecture 1
Lecture 2
Tutorials
Tutorial 1
Tutorial 2
References
Notebooks:
Lecture notebook (template)
Online tutorials and manuals:
Python Software Foundation, "The Python Tutorial" (Python 2 version, Python 3 version)
Python Software Foundation, "The Python Standard Library" (Python 2 version, Python 3 version)
Jupyter Team, "Notebook Examples"
The SciPy community, "NumPy User Guide"
The Matplotlib development team, "The Pyplot API"
Comment
Subscribe to: Posts by lhayward
My membership
Invite members
Get notifications
Broadcast message
Latest wiki comments
No comments have been made yet
Recent content
Programming in Python
Quantum Theory
psi 2019/Introduction to Mathematica
psi 2019/Math for QFT
psi 2019/Review of Classical Physics
Your Bookmarks in this Group
You have not yet bookmarked any content in this group. Click the "Bookmark this" link when viewing a piece of content to add it to this list.
Group members (46)
Managers:
Recent members:


jupyter psi2018-2019_programminginpython_template_2 (autosaved)
Programming in Python
PSI Front End
August 27-28, 2018
Lauren Hayward Sierens
0. Why learn Python?
1. Basics of Jupyter notebooks
You can run Python code using either a notebook (like this one) or a script. A script is executed from top to bottom, whereas a notebook is divided into cells. Here we use a notebook because its interactive environment and rich narrative text capabilities make it ideal for teaching and tutorial purposes.
In Jupyter notebooks, you type input into cells and execute a cell using Shift+Enter.
A cell can be either in Edit mode (green) or Command mode (blue).
1.0. Edit mode
Enter Edit mode by clicking on the cell's editor area or by pressing Enter or double clicking if the cell is in Command mode. In Edit mode, you can type into a cell to modify its contents. Use the Up and Down arrow keys to move between lines of the current cell.
1.1. Command mode
Enter Command mode by clicking on a cell outside of its editor area or by pressing Esc if the cell is in Edit mode. In Command mode, you can edit the notebook using commands such as:
Command
Description
y
make current cell a Code cell for writing and running code
m
make current cell a Markdown cell (like this one) for writing stylized text to document your notebook


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Programming in Python
PSI Front End
August 27-28, 2018
Lauren Hayward Sierens
0. Why learn Python?
• nice graphs and figures
• simulations
• data analysis
1. Basics of Jupyter notebooks
You can run Python code using either a notebook (like this one) or a script. A script is executed from top to bottom, whereas a notebook is divided into cells. Here we use a notebook because its interactive environment and rich narrative text capabilities make it ideal for teaching and tutorial purposes.
In Jupyter notebooks, you type input into cells and execute a cell using Shift+Enter.
A cell can be either in Edit mode (green) or Command mode (blue).
1.0. Edit mode
Enter Edit mode by clicking on the cell's editor area or by pressing Enter or double clicking if the cell is in Command mode. In Edit mode, you can type into a cell to modify its contents. Use the Up and Down arrow keys to move between lines of the current cell.
1.1. Command mode
Enter Command mode by clicking on a cell outside of its editor area or by pressing Esc if the cell is in Edit mode. In Command mode, you can edit the notebook using commands such as:
Command Description


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
Programming in Python
PSI Front End
August 27-28, 2018
Lauren Hayward Sierens
0. Why learn Python?
• nice graphs and figures
• simulations
• data analysis
• automating tasks
• readable and compact (complex operations often require only one line)
• lots of functionality available
1. Basics of Jupyter notebooks
You can run Python code using either a notebook (like this one) or a script. A script is executed from top to bottom, whereas a notebook is divided into cells. Here we use a notebook because its interactive environment and rich narrative text capabilities make it ideal for teaching and tutorial purposes.
In Jupyter notebooks, you type input into cells and execute a cell using Shift+Enter.
A cell can be either in Edit mode (green) or Command mode (blue).
1.0. Edit mode
Enter Edit mode by clicking on the cell's editor area or by pressing Enter or double clicking if the cell is in Command mode. In Edit mode, you can type into a cell to modify its contents. Use the Up and Down arrow keys to move between lines of the current cell.
1.1. Command mode
Enter Command mode by clicking on a cell outside of its editor area or by pressing Esc if the cell is in Edit mode. In Command mode, you can edit the notebook using commands such as:
Show All


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Programming in Python
PSI Front End
August 27-28, 2018
Lauren Hayward Sierens
0. Why learn Python?
* nice graphs and figures
* simulations
* data analysis
* automating tasks
* readable and compact (complex operations often require only one line)
* lots of functionality available
* extensible (integrates well with other languages)
* interpreted rather than compiled
1. Basics of Jupyter notebooks
You can run Python code using either a notebook (like this one) or a script. A script is executed from top to bottom, whereas a notebook is divided into cells. Here we use a notebook because its interactive environment and rich narrative text capabilities make it ideal for teaching and tutorial purposes.
In Jupyter notebooks, you type input into cells and execute a cell using Shift+Enter.
A cell can be either in Edit mode (green) or Command mode (blue).
1.0. Edit mode
Enter Edit mode by clicking on the cell's editor area or by pressing Enter or double clicking if the cell is in Command mode. In Edit mode, you can type into a cell to modify its contents. Use the Up and Down arrow keys to move between lines of the current cell.
1.1. Command mode


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
Programming in Python
PSI Front End
August 27-28, 2018
Lauren Hayward Sierens
0. Why learn Python?
• nice graphs and figures
• simulations
• data analysis
• automating tasks
• readable and compact (complex operations often require only one line)
• lots of functionality available
• extensible (integrates well with other languages)
• interpreted rather than compiled
• free
• good documentation
1. Basics of Jupyter notebooks
You can run Python code using either a notebook (like this one) or a script. A script is executed from top to bottom, whereas a notebook is divided into cells. Here we use a notebook because its interactive environment and rich narrative text capabilities make it ideal for teaching and tutorial purposes.
In Jupyter notebooks, you type input into cells and execute a cell using Shift+Enter.
A cell can be either in Edit mode (green) or Command mode (blue).
1.0. Edit mode
Enter Edit mode by clicking on the cell's editor area or by pressing Enter or double clicking if the cell is in Command mode. In Edit mode, you can type into a cell to modify its contents. Use the Up and Down arrow keys to move between lines of the current cell.
Show All

Programming in Python

PSI Front End

August 27-28, 2018

Lauren Hayward Sierens

0. Why learn Python?

- nice graphs and figures
- simulations
- data analysis
- automating tasks
- readable and compact (complex operations often require only one line)
- lots of functionality available
- extensible (integrates well with other languages)
- interpreted rather than compiled
- free
- good documentation
- machine learning
- scientific computing

1. Basics of Jupyter notebooks

You can run Python code using either a notebook (like this one) or a script. A script is executed from top to bottom, whereas a notebook is divided into cells. Here we use a notebook because its interactive environment and rich narrative text capabilities make it ideal for teaching and tutorial purposes.

In Jupyter notebooks, you type input into cells and execute a cell using `Shift+Enter`.

A cell can be either in *Edit mode* (green) or *Command mode* (blue).

1.0. Edit mode

Enter Edit mode by clicking on the cell's editor area or by pressing `Enter` or double clicking if the cell is in Command mode. In Edit mode, you can type into a

- interpreted rather than compiled
- free
- good documentation
- machine learning
- scientific computing

1. Basics of Jupyter notebooks

You can run Python code using either a notebook (like this one) or a script. A script is executed from top to bottom, whereas a notebook is divided into cells. Here we use a notebook because its interactive environment and rich narrative text capabilities make it ideal for teaching and tutorial purposes.

In Jupyter notebooks, you type input into cells and execute a cell using `Shift+Enter`.

A cell can be either in *Edit mode* (green) or *Command mode* (blue).

1.0. Edit mode

Enter Edit mode by clicking on the cell's editor area or by pressing `Enter` or double clicking if the cell is in Command mode. In Edit mode, you can type into a cell to modify its contents. Use the `Up` and `Down` arrow keys to move between lines of the current cell.

1.1. Command mode ¶

Enter Command mode by clicking on a cell outside of its editor area or by pressing `Esc` if the cell is in Edit mode. In Command mode, you can edit the notebook using commands such as:

Command	Description
`Y`	make current cell a Code cell for writing and running code
`M`	make current cell a Markdown cell (like this one) for writing stylized text to document your notebook
`Up`	move to the cell above the current cell
`Down`	move to the cell below the current cell
`A`	insert cell above current cell
`B`	insert cell below current cell
`D+D`	delete current cell
`C`	copy current cell

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Up	move to the cell above the current cell
Down	move to the cell below the current cell
A	insert cell above current cell
B	insert cell below current cell
D+D	delete current cell
C	copy current cell
X	cut current cell
V	paste cell below current cell
Shift+V	paste cell above current cell
F	open find and replace for current cell

To see the full list of command-mode commands, open the *command palette* above by clicking on the picture of the keyboard or by pressing `Command+Shift+P` (Mac) or `Ctrl+Shift+P` (Windows).

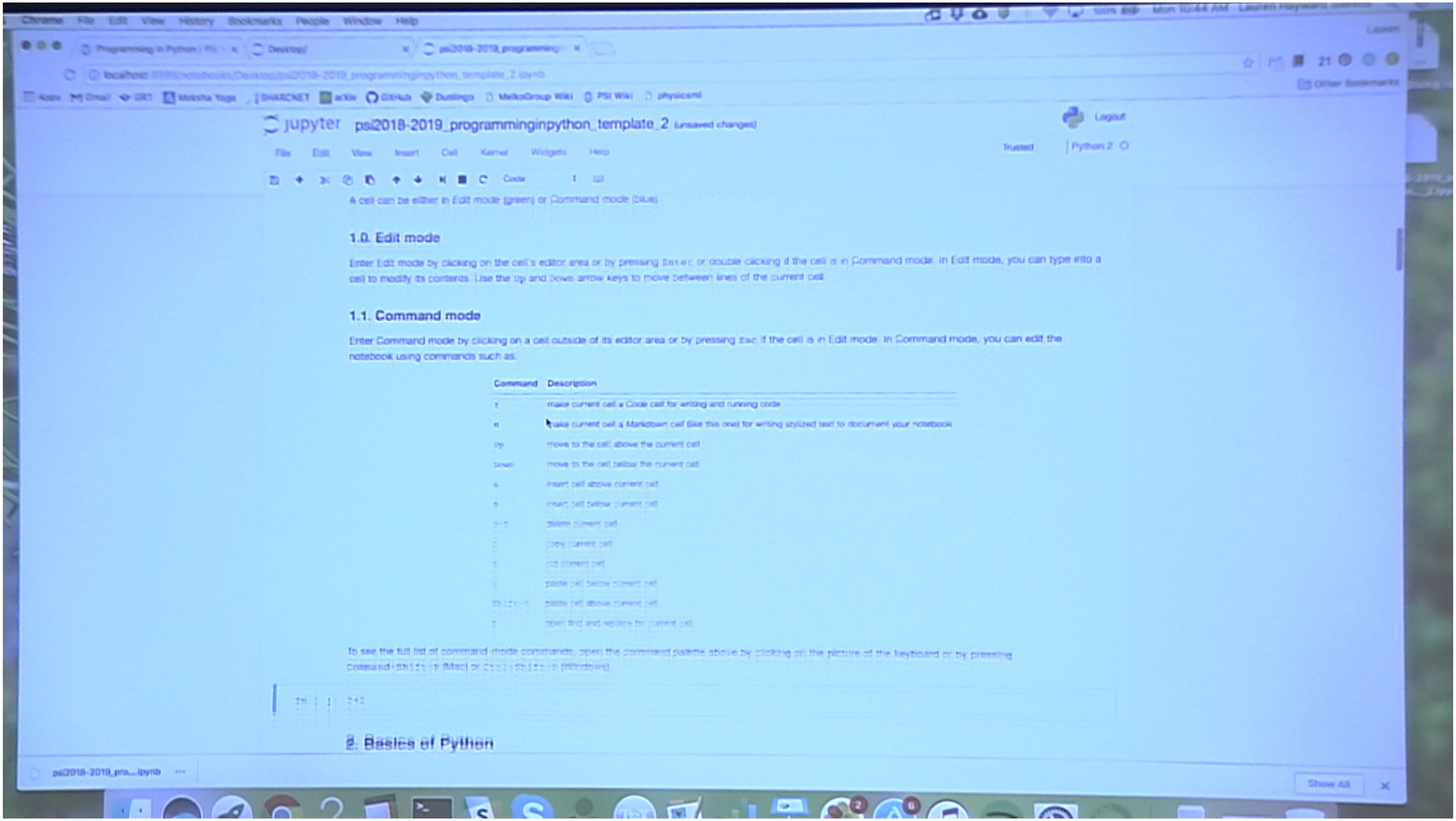
```
In [1]: 2+2
Out[1]: 4
```

2. Basics of Python

2.0. Numbers

We can use Python as a calculator for evaluating numerical expressions. Available arithmetic operators are:

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division or floor division (see below)
//	floor division (also known as integer division)
**	exponent (**Note:** the operator ^ is not an exponential operator!)


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
A cell can be either in Edit mode (green) or Command mode (blue)
1.0. Edit mode
Enter Edit mode by clicking on the cell's editor area or by pressing Enter or double clicking if the cell is in Command mode. In Edit mode, you can type into a cell to modify its contents. Use the Up and Down arrow keys to move between lines of the current cell.
1.1. Command mode
Enter Command mode by clicking on a cell outside of its editor area or by pressing Esc if the cell is in Edit mode. In Command mode, you can edit the notebook using commands such as:
Command Description
Y make current cell a Code cell for writing and running code
M make current cell a Markdown cell (like this one) for writing stylized text to document your notebook
Up move to the cell above the current cell
Down move to the cell below the current cell
A insert cell above current cell
B insert cell below current cell
D+D delete current cell
C copy current cell
X cut current cell
V paste cell below current cell
Shift+V paste cell above current cell
F open find and replace for current cell
To see the full list of command mode commands, open the command palette above by clicking on the picture of the keyboard or by pressing Command+Shift+P (Mac) or Ctrl+Shift+P (Windows)
2. Basics of Python

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

`V`	paste cell below current cell
`Shift+V`	paste cell above current cell
`F`	open find and replace for current cell

To see the full list of command-mode commands, open the *command palette* above by clicking on the picture of the keyboard or by pressing `Command+Shift+P` (Mac) or `Ctrl+Shift+P` (Windows).

```
In [ ]:
```

2. Basics of Python

2.0. Numbers

We can use Python as a calculator for evaluating numerical expressions. Available arithmetic operators are:

Operator	Description
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division or floor division (see below)
`//`	floor division (also known as integer division)
`**`	exponent (**Note:** the operator `^` is not an exponential operator!)
`%`	modulus (returns the remainder upon dividing the left-hand operand by the right-hand operand)

```
In [ ]:
```

Numbers can have different numeric types. Integer numbers (such as `2`, `-9` and `0`) have type `int`. Numbers with a decimal (such as `13.4`, `4.0` and `-12.2`) have type `float`. You can also read about long integers and complex numbers, but we won't use them in this course. Python can determine the number's type without being explicitly told.

Booleans (variables that take one of two possible values) are a subtype of integers.

```
In [ ]:
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

`V`	paste cell below current cell
`Shift+V`	paste cell above current cell
`F`	open find and replace for current cell

To see the full list of command-mode commands, open the *command palette* above by clicking on the picture of the keyboard or by pressing `Command+Shift+P` (Mac) or `Ctrl+Shift+P` (Windows).

```
In [ ]:
```

2. Basics of Python

2.0. Numbers

We can use Python as a calculator for evaluating numerical expressions. Available arithmetic operators are:

Operator	Description
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division or floor division (see below)
`//`	floor division (also known as integer division)
`**`	exponent (**Note:** the operator `^` is not an exponential operator!)
`%`	modulus (returns the remainder upon dividing the left-hand operand by the right-hand operand)

```
In [2]: (1+3)*(6-4)**3
Out[2]: 32
```

Numbers can have different numeric types. Integer numbers (such as `2`, `-9` and `0`) have type `int`. Numbers with a decimal (such as `13.4`, `4.0` and `-12.2`) have type `float`. You can also read about long integers and complex numbers, but we won't use them in this course. Python can determine the number's type without being explicitly told.

Booleans (variables that take one of two possible values) are a subtype of integers.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division or floor division (see below)
//	floor division (also known as integer division)
**	exponent (**Note:** the operator ^ is not an exponential operator!)
%	modulus (returns the remainder upon dividing the left-hand operand by the right-hand operand)

```
In [2]: (1+3)*(6-4)**3

Out[2]: 32

In [ ]: (%)
```

Numbers can have different numeric types. Integer numbers (such as `2`, `-9` and `0`) have type `int`. Numbers with a decimal (such as `13.4`, `4.0` and `-12.2`) have type `float`. You can also read about long integers and complex numbers, but we won't use them in this course. Python can determine the number's type without being explicitly told.

Booleans (variables that take one of two possible values) are a subtype of integers.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division or floor division (see below)
//	floor division (also known as integer division)
**	exponent (**Note:** the operator ^ is not an exponential operator!)
%	modulus (returns the remainder upon dividing the left-hand operand by the right-hand operand)

```
In [2]: (1+3)*(6-4)**3
Out[2]: 32
```

```
In [3]: 9%5
Out[3]: 4
```

Numbers can have different numeric types. Integer numbers (such as `2`, `-9` and `0`) have type `int`. Numbers with a decimal (such as `13.4`, `4.0` and `-12.2`) have type `float`. You can also read about long integers and complex numbers, but we won't use them in this course. Python can determine the number's type without being explicitly told.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

`**` exponent (**Note:** the operator `^` is not an exponential operator!)

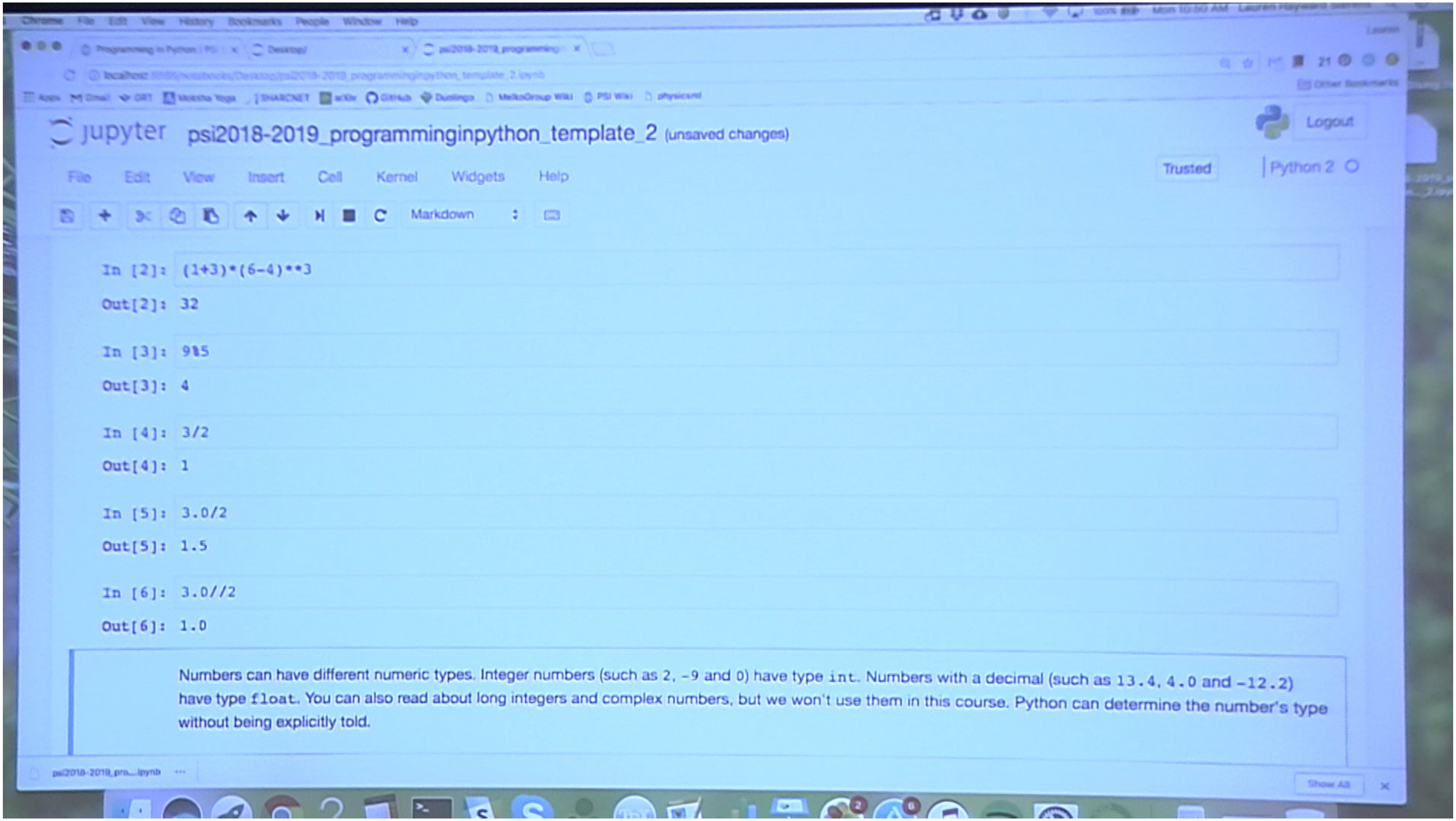
`%` modulus (returns the remainder upon dividing the left-hand operand by the right-hand operand)

```
In [2]: (1+3)*(6-4)**3
Out[2]: 32

In [3]: 9%5
Out[3]: 4

In [4]: 3/2
Out[4]: 1
```

Numbers can have different numeric types. Integer numbers (such as `2`, `-9` and `0`) have type `int`. Numbers with a decimal (such as `13.4`, `4.0` and `-12.2`) have type `float`. You can also read about long integers and complex numbers, but we won't use them in this course. Python can determine the number's type without being explicitly told.

Booleans (variables that take one of two possible values) are a subtype of integers.

```
In [ ]:
```


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
In [2]: (1+3)*(6-4)**3
Out[2]: 32
In [3]: 9%5
Out[3]: 4
In [4]: 3/2
Out[4]: 1
In [5]: 3.0/2
Out[5]: 1.5
In [6]: 3.0//2
Out[6]: 1.0
Numbers can have different numeric types. Integer numbers (such as 2, -9 and 0) have type int. Numbers with a decimal (such as 13.4, 4.0 and -12.2) have type float. You can also read about long integers and complex numbers, but we won't use them in this course. Python can determine the number's type without being explicitly told.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

In [ ]:

2.1. Printing

If we enter several numerical expressions on different lines of the same cell, only the last line will be displayed.

In [7]:
```
2+2
4-6
```
Out[7]: -2

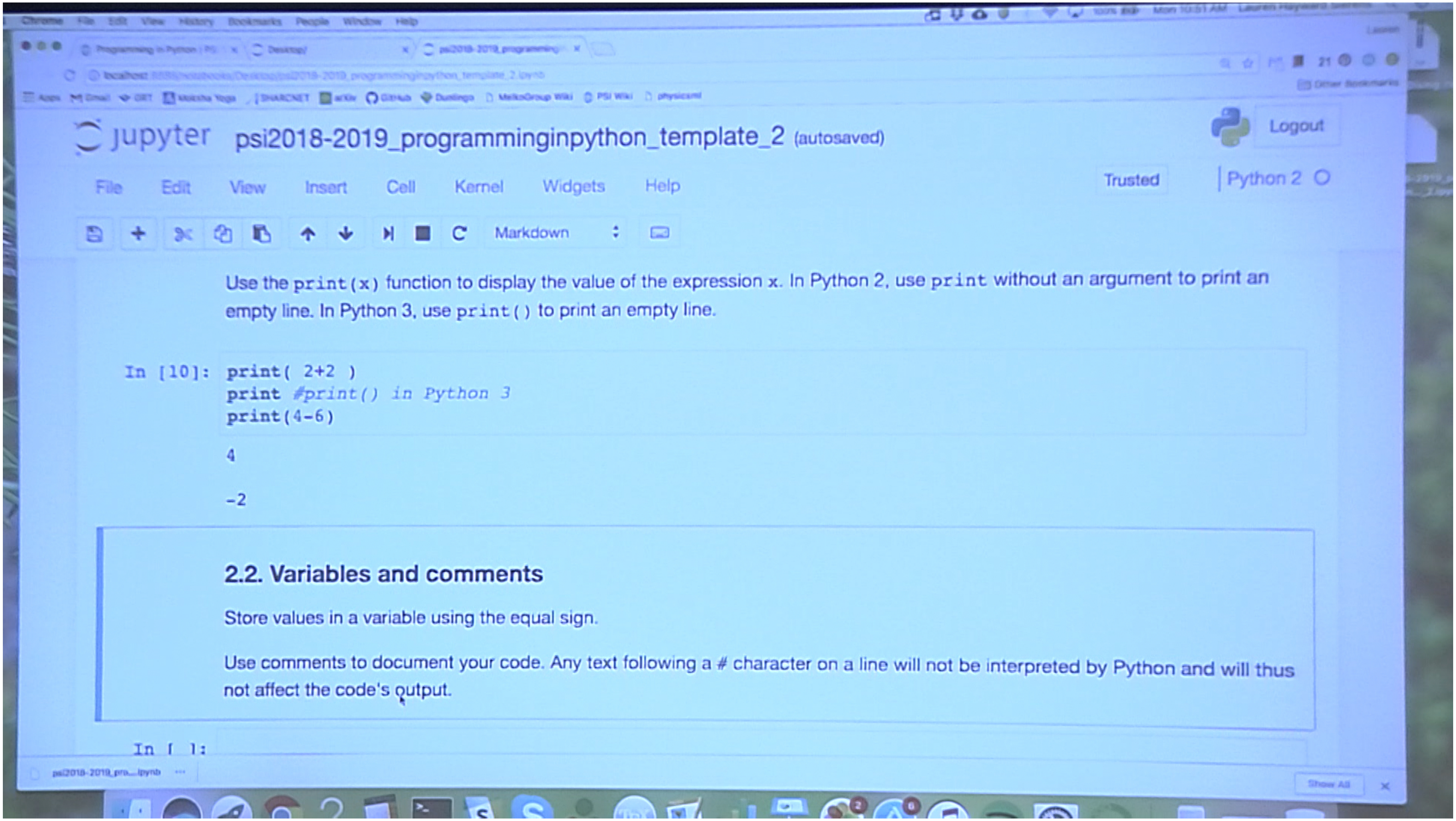
Use the `print(x)` function to display the value of the expression `x`. In Python 2, use `print` without an argument to print an empty line. In Python 3, use `print()` to print an empty line.

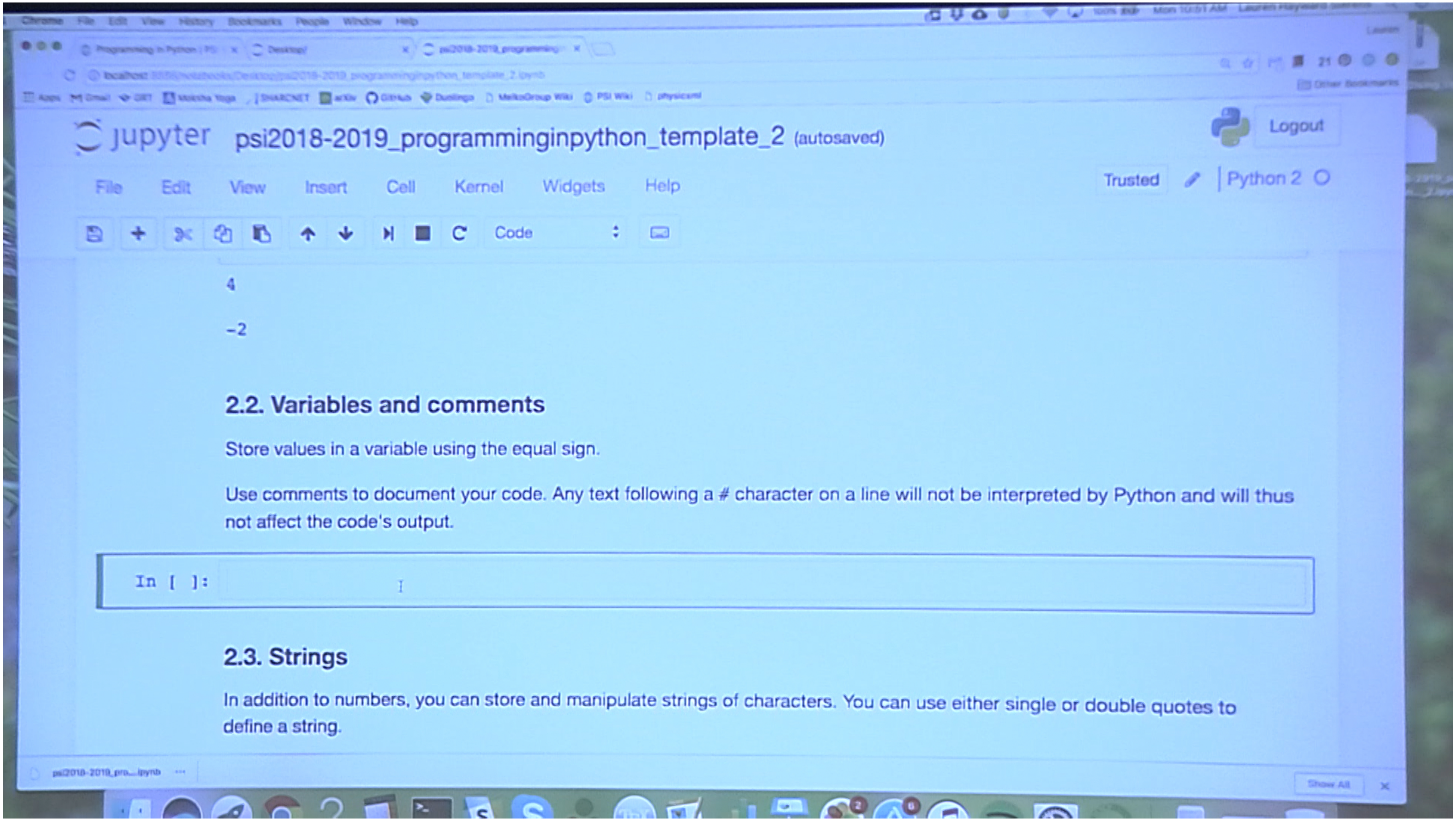
In [ ]:

2.2. Variables and comments

Store values in a variable using the equal sign.

Use comments to document your code. Any text following a # character on a line will not be interpreted by Python and will thus not affect the code's output.


jupyter psi2018-2019_programminginpython_template_2 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
Use the print(x) function to display the value of the expression x. In Python 2, use print without an argument to print an empty line. In Python 3, use print() to print an empty line.
In [10]:
print(2+2)
print #print() in Python 3
print(4-6)
4
-2
2.2. Variables and comments
Store values in a variable using the equal sign.
Use comments to document your code. Any text following a # character on a line will not be interpreted by Python and will thus not affect the code's output.
In [ ]:


jupyter psi2018-2019_programminginpython_template_2 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
4
-2
2.2. Variables and comments
Store values in a variable using the equal sign.
Use comments to document your code. Any text following a # character on a line will not be interpreted by Python and will thus not affect the code's output.
In [ ]:
2.3. Strings
In addition to numbers, you can store and manipulate strings of characters. You can use either single or double quotes to define a string.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 2

Code

```
h = 5 #height
print( w*h )
```

50

2.3. Strings

In addition to numbers, you can store and manipulate strings of characters. You can use either single or double quotes to define a string.

In [ ]:

Use the format method to get formatted strings, which are often useful for displaying output in a nice way

In [ ]:

Exercise: Write code that takes a variable storing a circle's radius and prints its area. For example, when `r=5`, you should print:

```
The area of the circle is 78.53975.
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

```
h = 5 #height
print( w*h )
```

```
50
```

2.3. Strings

In addition to numbers, you can store and manipulate strings of characters. You can use either single or double quotes to define a string.

In [12]:

```
word1 = "Hello"
word2 = "world"
sentence = word1 + word2 #concatenate using '+'
print(sentence)
```

```
Helloworld
```

Use the format method to get formatted strings, which are often useful for displaying output in a nice way

In [ ]:

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 2

Code

```
h = 5 #height
print( w*h )
```

```
50
```

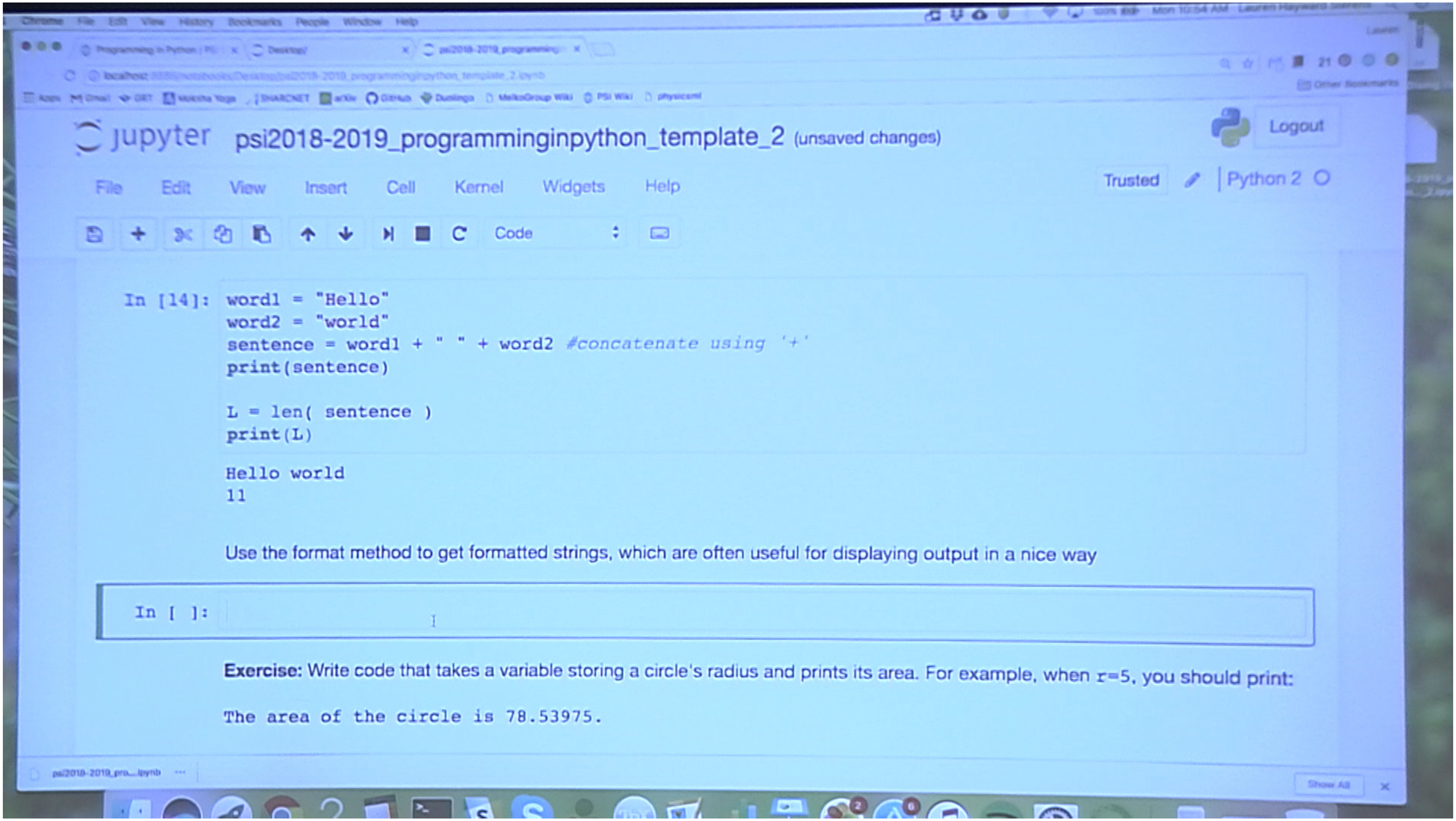
2.3. Strings

In addition to numbers, you can store and manipulate strings of characters. You can use either single or double quotes to define a string.

In [13]:
```
word1 = "Hello"
word2 = "world"
sentence = word1 + " " + word2 #concatenate using '+'
print(sentence)

L = len()
```

```
Hello world
```

Use the format method to get formatted strings, which are often useful for displaying output in a nice way


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
In [14]:
word1 = "Hello"
word2 = "world"
sentence = word1 + " " + word2 #concatenate using '+'
print(sentence)
L = len(sentence)
print(L)
Hello world
11
Use the format method to get formatted strings, which are often useful for displaying output in a nice way
In [ ]:
Exercise: Write code that takes a variable storing a circle's radius and prints its area. For example, when r=5, you should print:
The area of the circle is 78.53975.

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Notebook saved | Trusted | Python 2

In [14]:
```
word1 = "Hello"
word2 = "world"
sentence = word1 + " " + word2 #concatenate using '+'
print(sentence)

L = len( sentence )
print(L)
```

```
Hello world
11
```

Use the format method to get formatted strings, which are often useful for displaying output in a nice way

In [ ]:
```
name = "Lauren"
x = 0
y = 3.2
print( "The coordinates of {} are ({},{})".format(name,x,y) )
```

Exercise: Write code that takes a variable storing a circle's radius and prints its area. For example, when `r=5`, you should print:

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

```
print(sentence)

L = len( sentence )
print(L)
```

```
Hello world
11
```

Use the format method to get formatted strings, which are often useful for displaying output in a nice way

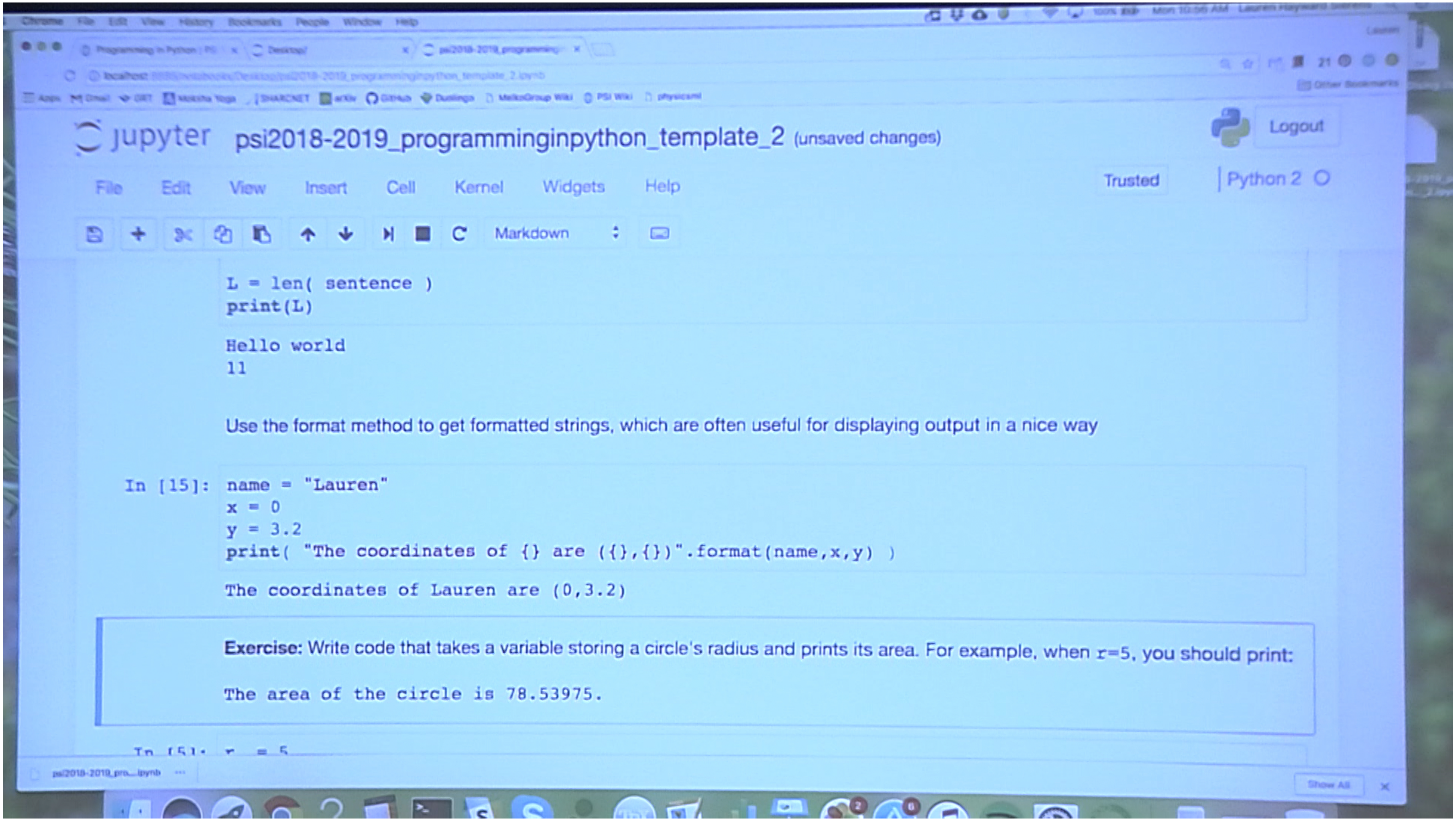
In [15]:

```
name = "Lauren"
x = 0
y = 3.2
print( "The coordinates of {} are ({},{})".format(name,x,y) )
```

```
The coordinates of Lauren are (0,3.2)
```

Exercise: Write code that takes a variable storing a circle's radius and prints its area. For example, when `r=5`, you should print:

`The area of the circle is 78.53975.`


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
L = len(sentence)
print(L)
Hello world
11
Use the format method to get formatted strings, which are often useful for displaying output in a nice way
In [15]:
name = "Lauren"
x = 0
y = 3.2
print("The coordinates of {} are ({},{})".format(name,x,y))
The coordinates of Lauren are (0,3.2)
Exercise: Write code that takes a variable storing a circle's radius and prints its area. For example, when r=5, you should print:
The area of the circle is 78.53975.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Use the format method to get formatted strings, which are often useful for displaying output in a nice way

In [15]:
```
name = "Lauren"
x = 0
y = 3.2
print( "The coordinates of {} are ({},{})".format(name,x,y) )
```

```
The coordinates of Lauren are (0,3.2)
```

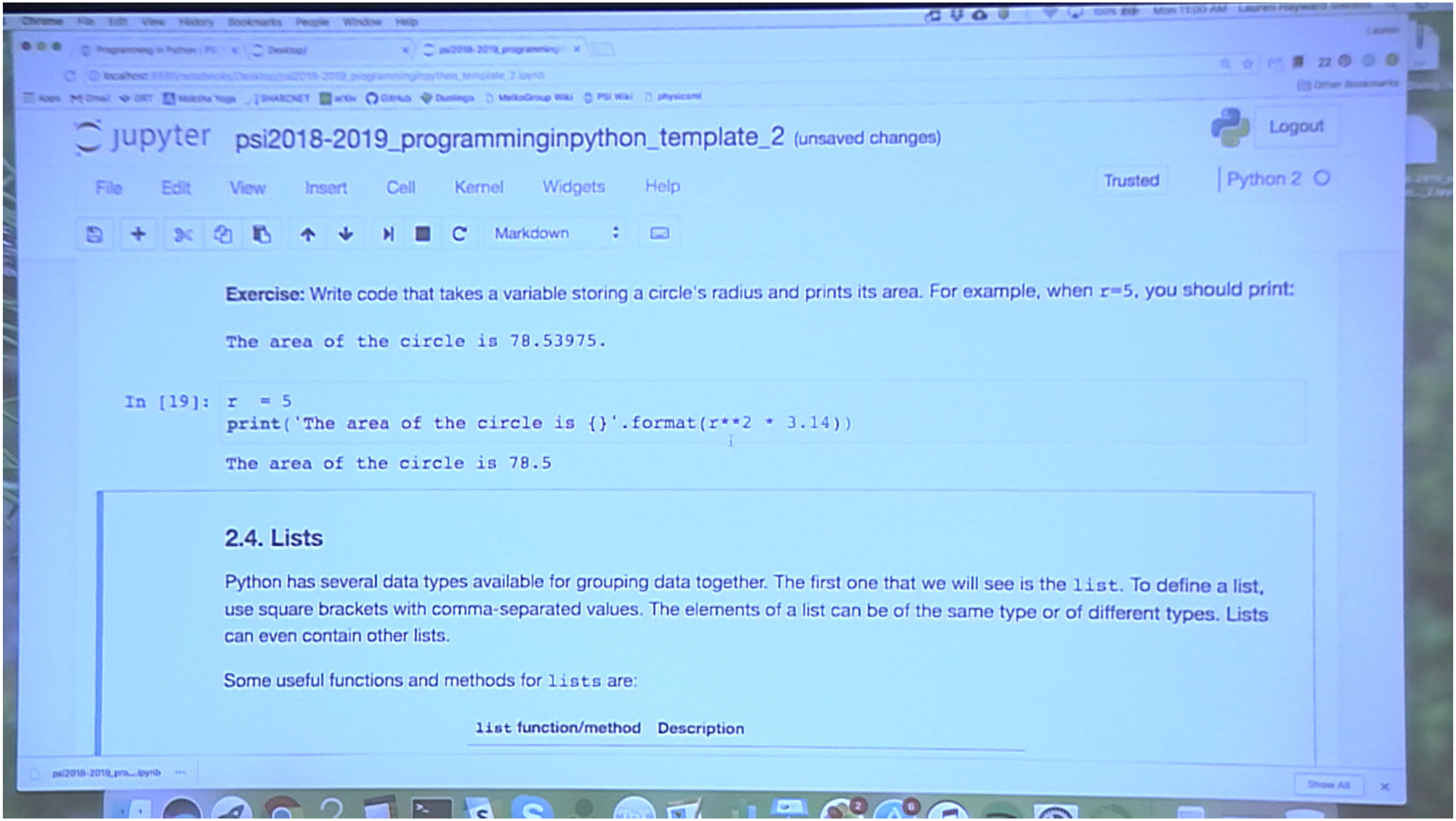
Exercise: Write code that takes a variable storing a circle's radius and prints its area. For example, when `r=5`, you should print:

```
The area of the circle is 78.53975.
```

In [5]:
```
r  = 5
```

2.4. Lists

Python has several data types available for grouping data together. The first one that we will see is the `list`. To define a list,


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
Exercise: Write code that takes a variable storing a circle's radius and prints its area. For example, when r=5, you should print:
The area of the circle is 78.53975.
In [19]: r = 5
print('The area of the circle is {}'.format(r**2 * 3.14))
The area of the circle is 78.5
2.4. Lists
Python has several data types available for grouping data together. The first one that we will see is the list. To define a list, use square brackets with comma-separated values. The elements of a list can be of the same type or of different types. Lists can even contain other lists.
Some useful functions and methods for lists are:
list function/method Description

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

Python has several data types available for grouping data together. The first one that we will see is the `list`. To define a list, use square brackets with comma-separated values. The elements of a list can be of the same type or of different types. Lists can even contain other lists.

Some useful functions and methods for `lists` are:

`list` function/method	Description
`+`	concatenates `lists`
`len(list)`	outputs the `list` length
`min(list)`	outputs the smallest item in `list`
`max(list)`	outputs the largest item in `list`
`list.append(x)`	adds a new item `x` to the end of `list`
`list.sort()`	sorts the elements of `list`
`sorted(list)`	return a new sorted list with the elements of `list`

In [ ]:

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

```
print( A )
print( B )
```

```
[4, 12, -2]
['a', 0, 4.2, [4, 5]]
```

In [22]:
```
A+B
```

Out[22]:
```
[4, 12, -2, 'a', 0, 4.2, [4, 5]]
```

In [23]:
```
max(A)
```

Out[23]:
```
12
```

2.5. Indexing and slicing

Both strings and lists can be indexed. The first element always has index 0.

You can also use negative indices to index elements counting from end.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

In [22]:
```
A+B
```
Out[22]:
```
[4, 12, -2, 'a', 0, 4.2, [4, 5]]
```

In [23]:
```
max(A)
```
Out[23]:
```
12
```

In [26]:
```
A.append(3)
print( A )

A.append(B)
print( A )
```
```
[4, 12, -2, 3, 3, ['a', 0, 4.2, [4, 5]], 3]
[4, 12, -2, 3, 3, ['a', 0, 4.2, [4, 5]], 3, ['a', 0, 4.2, [4, 5]]]
```

2.5. Indexing and slicing

Both strings and lists can be indexed. The first element always has index 0.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

In [30]:
```
A.append(3)
print( A )

A.append(B)
print( A )
```

```
[4, 12, -2, 3]
[4, 12, -2, 3, ['a', 0, 4.2, [4, 5]]]
```

In [32]:
```
C = [6,-2.45, 12.3, -203]
print( C )
D = sorted(C)
print( D )
```

```
[6, -2.45, 12.3, -203]
[-203, -2.45, 6, 12.3]
```

2.5. Indexing and slicing

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

```
[6, -2.45, 12.3, -203]

[6, -2.45, 12.3, -203]
[-203, -2.45, 6, 12.3]

[-203, -2.45, 6, 12.3]
```

2.5. Indexing and slicing

Both strings and lists can be indexed. The first element always has index 0.

You can also use negative indices to index elements counting from end.

```
In [6]: B = ["a", 0, 4.2, [4,5] ]
```

Use *slicing* to get a subset of your list or string.

```
In [7]: s = "abcdef"
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

```
[6, -2.45, 12.3, -203]
[-203, -2.45, 6, 12.3]

[-203, -2.45, 6, 12.3]
```

2.5. Indexing and slicing

Both strings and lists can be indexed. The first element always has index 0.

You can also use negative indices to index elements counting from end.

```
In [38]: B = ["a", 0, 4.2, [4,5] ]

         print( B[0] ) #print the first character
         print( B[-1] ) #print the last character
         print( B[-4] ) #print the fourthnlast character

a
[4, 5]
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

2.5. Indexing and slicing

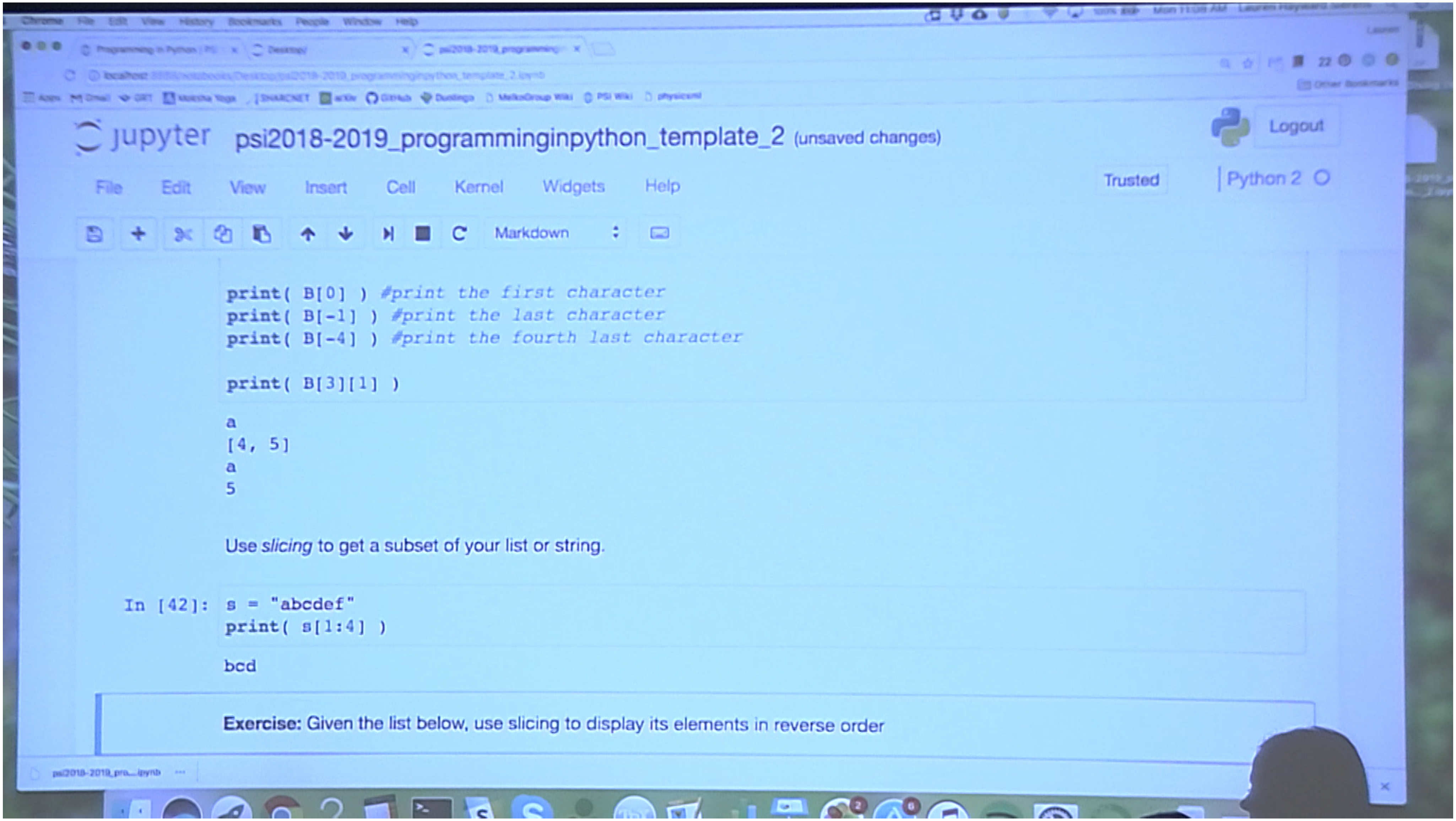
Both strings and lists can be indexed. The first element always has index 0.

You can also use negative indices to index elements counting from end.

In [41]:
```
B = ["a", 0, 4.2, [4,5] ]

print( B[0] ) #print the first character
print( B[-1] ) #print the last character
print( B[-4] ) #print the fourth last character

print( B[3][1] )
```

```
a
[4, 5]
a
5
```

Use *slicing* to get a subset of your list or string.


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
print(B[0]) #print the first character
print(B[-1]) #print the last character
print(B[-4]) #print the fourth last character
print(B[3][1])
a
[4, 5]
a
5
Use slicing to get a subset of your list or string.
In [42]: s = "abcdef"
print(s[1:4])
bcd
Exercise: Given the list below, use slicing to display its elements in reverse order

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

```
print( B[-4] ) #print the fourth last character

print( B[3][1] )
```

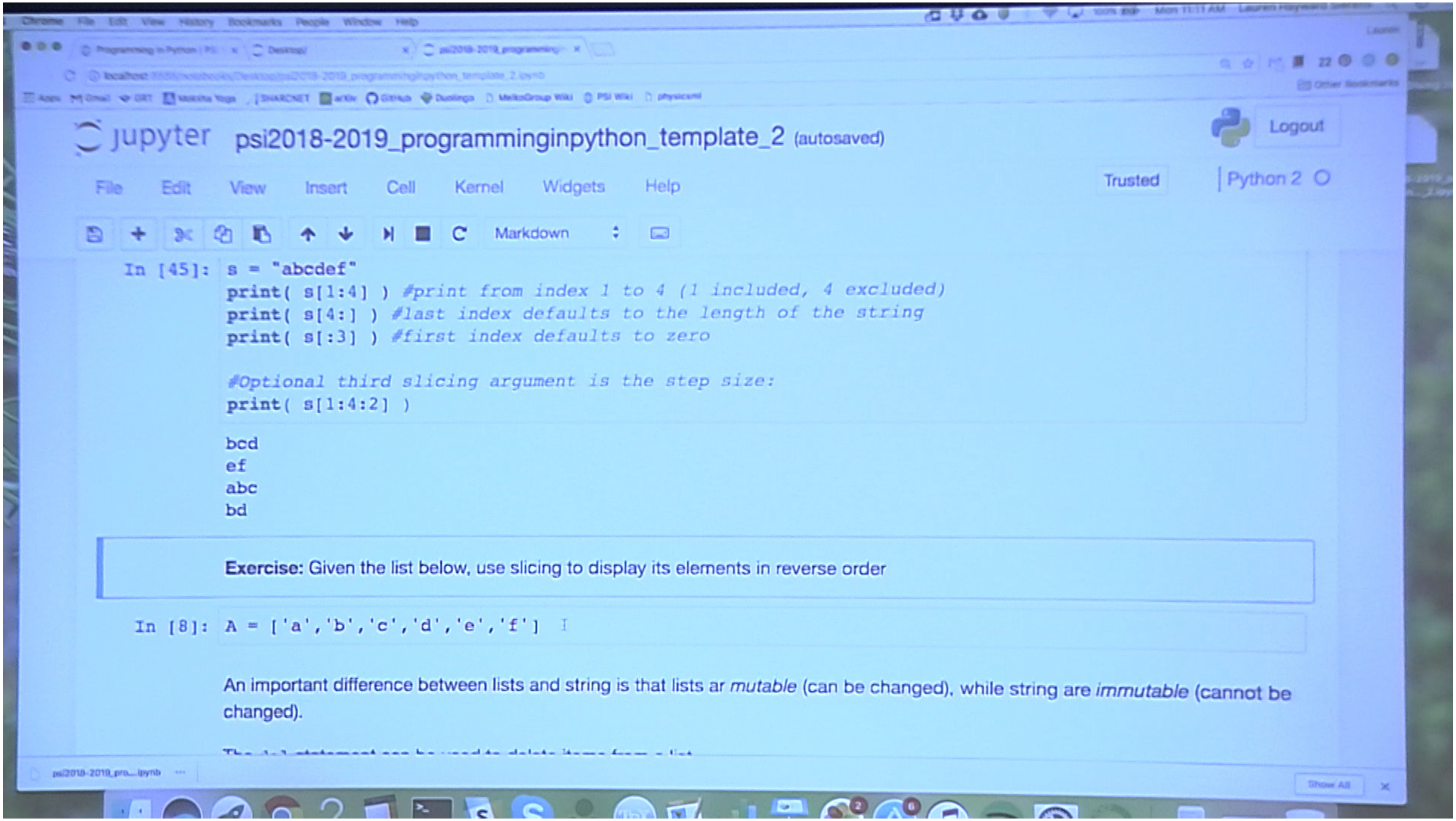
```
a
[4, 5]
a
5
```

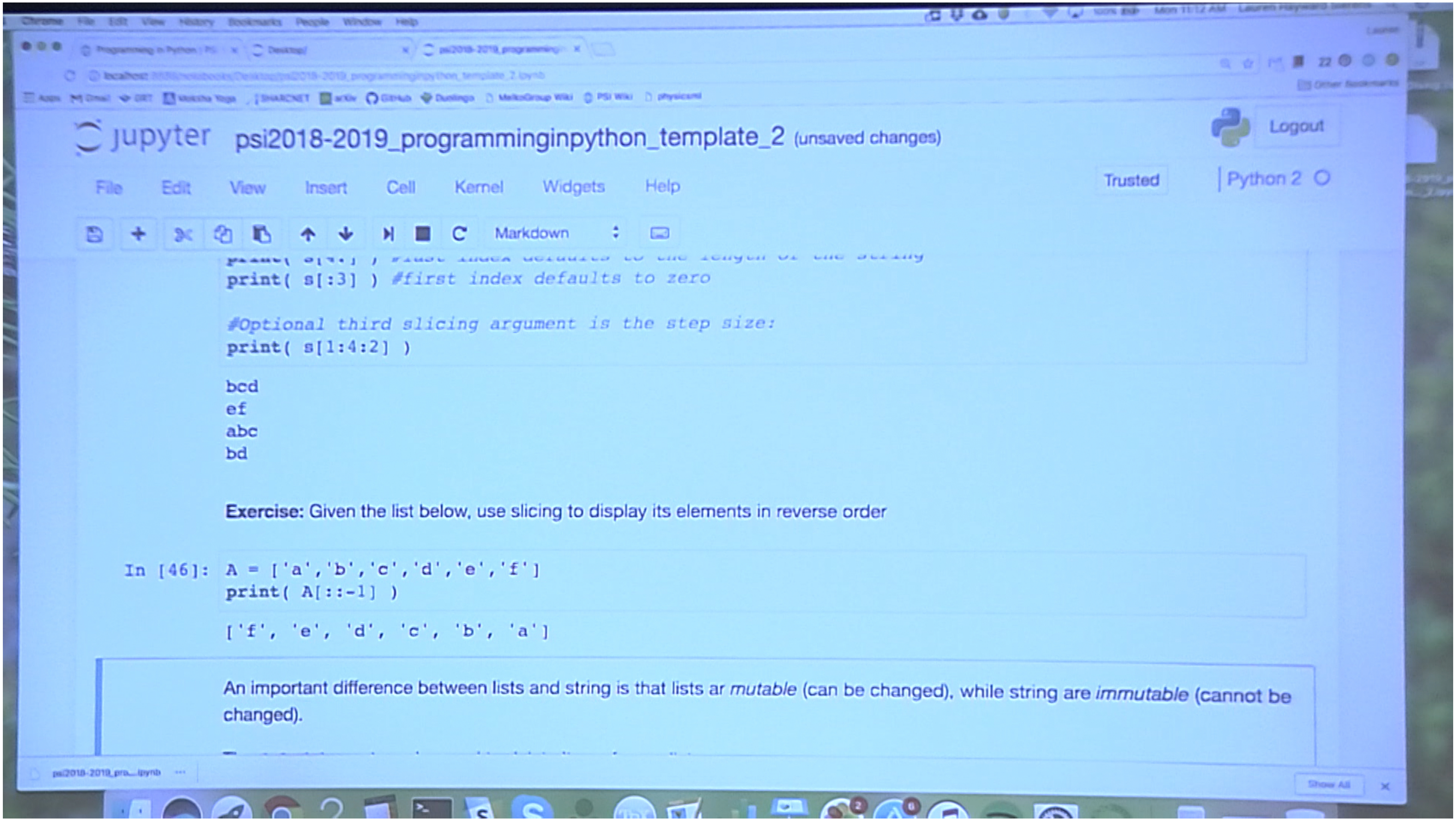
Use *slicing* to get a subset of your list or string.

In [44]:
```
s = "abcdef"
print( s[1:4] ) #print from index 1 to 4 (1 included, 4 excluded)
print( s[4:] ) #last index defaults to the length of the string
print( s[:3] ) #first index defaults to zero
```

```
bcd
ef
abc
```

Exercise: Given the list below, use slicing to display its elements in reverse order


jupyter psi2018-2019_programminginpython_template_2 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
In [45]: s = "abcdef"
print(s[1:4]) #print from index 1 to 4 (1 included, 4 excluded)
print(s[4:]) #last index defaults to the length of the string
print(s[:3]) #first index defaults to zero
#Optional third slicing argument is the step size:
print(s[1:4:2])
bcd
ef
abc
bd
Exercise: Given the list below, use slicing to display its elements in reverse order
In [8]: A = ['a','b','c','d','e','f']
An important difference between lists and string is that lists ar mutable (can be changed), while string are immutable (cannot be changed).



jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
print(s[:3]) #first index defaults to zero
#Optional third slicing argument is the step size:
print(s[1:4:2])
bcd
ef
abc
bd
Exercise: Given the list below, use slicing to display its elements in reverse order
In [46]: A = ['a','b','c','d','e','f']
print(A[::-1])
['f', 'e', 'd', 'c', 'b', 'a']
An important difference between lists and string is that lists ar mutable (can be changed), while string are immutable (cannot be changed).

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

```
print( A[::-1] )
```

```
['f', 'e', 'd', 'c', 'b', 'a']
```

An important difference between lists and string is that lists ar *mutable* (can be changed), while string are *immutable* (cannot be changed).

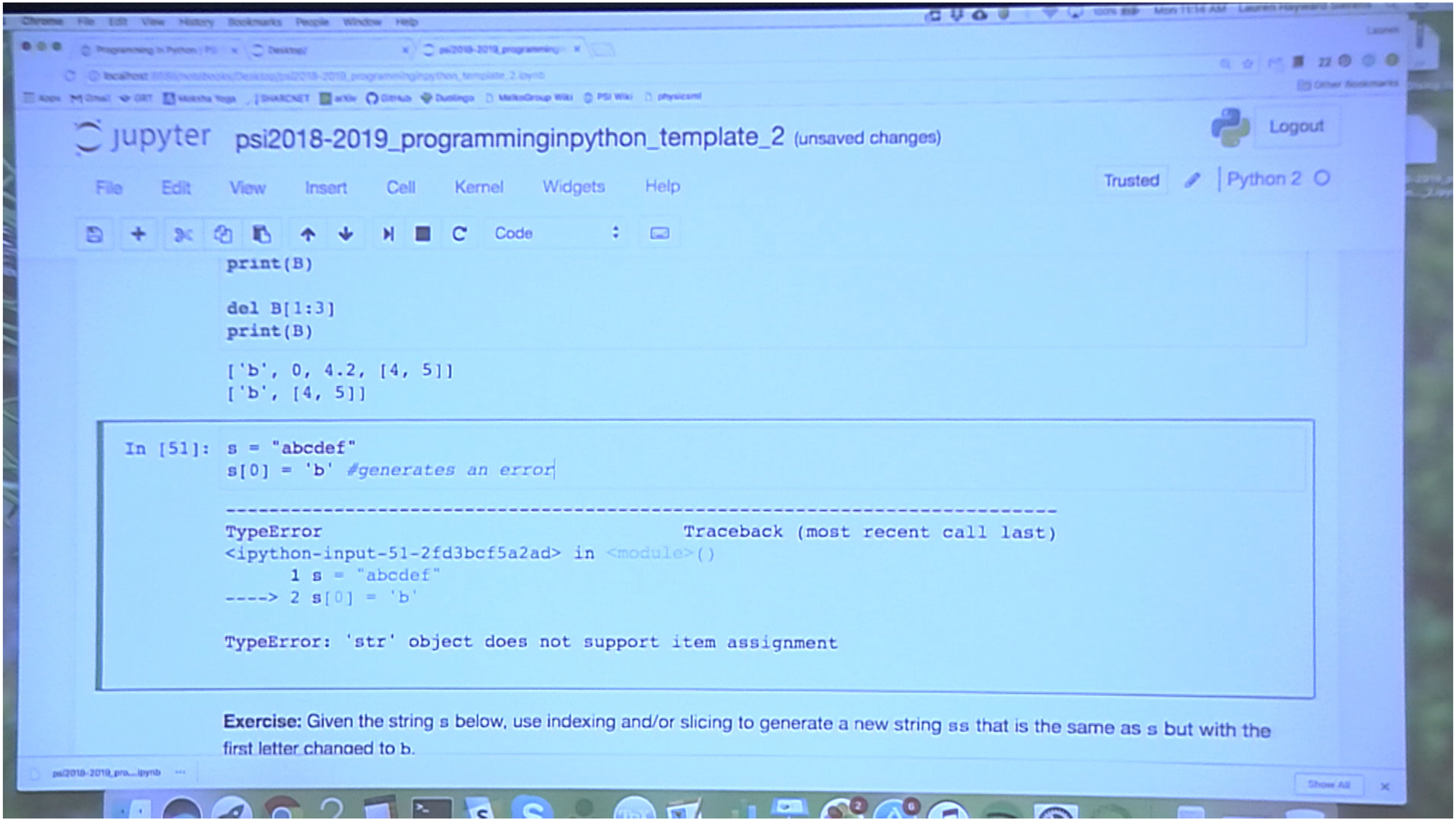
The `del` statement can be used to delete items from a list.

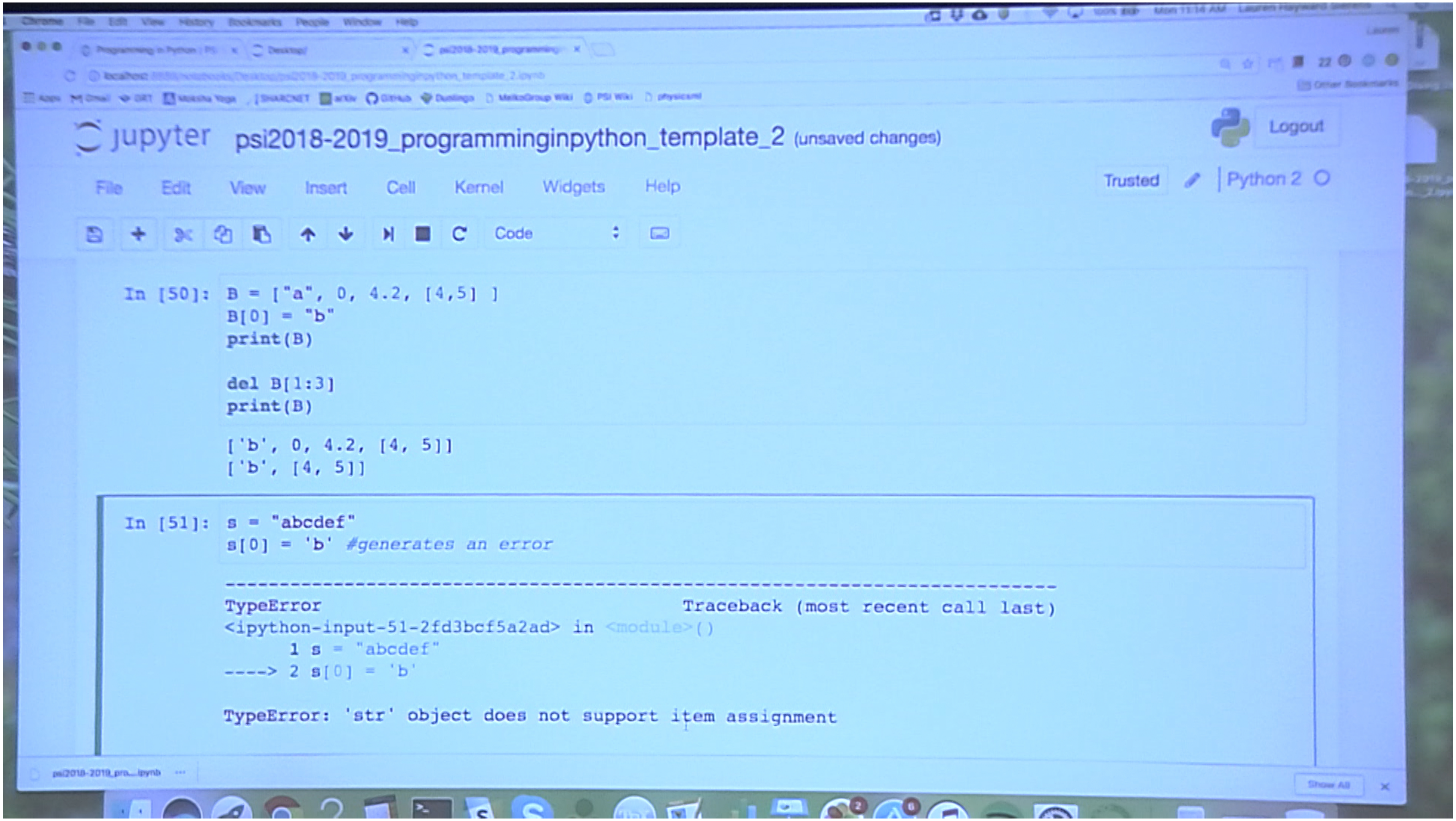
```
In [9]: B = ["a", 0, 4.2, [4,5] ]
```

```
In [10]: s = "abcdef"
```

Exercise: Given the string `s` below, use indexing and/or slicing to generate a new string `ss` that is the same as `s` but with the first letter changed to `b`.

```
In [11]: s = "abcdef"
```


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
print(B)
del B[1:3]
print(B)
['b', 0, 4.2, [4, 5]]
['b', [4, 5]]
In [51]: s = "abcdef"
s[0] = 'b' #generates an error
TypeError Traceback (most recent call last)
<ipython-input-51-2fd3bcf5a2ad> in <module>()
1 s = "abcdef"
----> 2 s[0] = 'b'
TypeError: 'str' object does not support item assignment
Exercise: Given the string s below, use indexing and/or slicing to generate a new string ss that is the same as s but with the
first letter changed to b.


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
In [50]: B = ["a", 0, 4.2, [4,5]]
B[0] = "b"
print(B)
del B[1:3]
print(B)
['b', 0, 4.2, [4, 5]]
['b', [4, 5]]
In [51]: s = "abcdef"
s[0] = 'b' #generates an error

TypeError Traceback (most recent call last)
<ipython-input-51-2fd3bcf5a2ad> in <module>()
1 s = "abcdef"
----> 2 s[0] = 'b'
TypeError: 'str' object does not support item assignment

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Notebook saved Trusted Python 2

first letter changed to `b`.

```
In [11]: s = "abcdef"
```

2.6. Dictionaries

Dictionaries are another mutable data type in Python. Unlike a `list`, which is indexed by a range of numbers, a `dict` is indexed by keys that you define. Each key can be a string, a number, or any other immutable type. You can think of a dictionary as a mapping.

```
In [ ]: D
```

You can read about other available data structures such as tuples and sets, but we won't use them in this course.

2.7. Other useful Python tools

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

first letter changed to b.

```
In [11]: s = "abcdef"
```

2.6. Dictionaries

Dictionaries are another mutable data type in Python. Unlike a `list`, which is indexed by a range of numbers, a `dict` is indexed by keys that you define. Each key can be a string, a number, or any other immutable type. You can think of a dictionary as a mapping.

```
In [53]: D = { "abc":"a", "a":0, 4:2 }
         print(D)

{'a': 0, 'abc': 'a', 4: 2}
```

You can read about other available data structures such as tuples and sets, but we won't use them in this course.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 2

Markdown

first letter changed to b.

In [11]:
```
s = "abcdef"
```

2.6. Dictionaries

Dictionaries are another mutable data type in Python. Unlike a `list`, which is indexed by a range of numbers, a `dict` is indexed by keys that you define. Each key can be a string, a number, or any other immutable type. You can think of a dictionary as a mapping.

In [55]:
```
D = { "abc":"a", "a":0, 4:2 }
print(D)
print( D["a"] )
print( D.keys() )
```

```
{'a': 0, 'abc': 'a', 4: 2}
0
['a', 'abc', 4]
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 2

Code

first letter changed to b.

```
In [11]: s = "abcdef"
```

2.6. Dictionaries

Dictionaries are another mutable data type in Python. Unlike a `list`, which is indexed by a range of numbers, a `dict` is indexed by keys that you define. Each key can be a string, a number, or any other immutable type. You can think of a dictionary as a mapping.

```
In [55]: D = { "abc":"a", "a":0, 4:2 }
         print(D)
         print( D["a"] )
         print( D.keys() )

         D["new key"]

{'a': 0, 'abc': 'a', 4: 2}
0
```

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Notebook saved Trusted Python 2

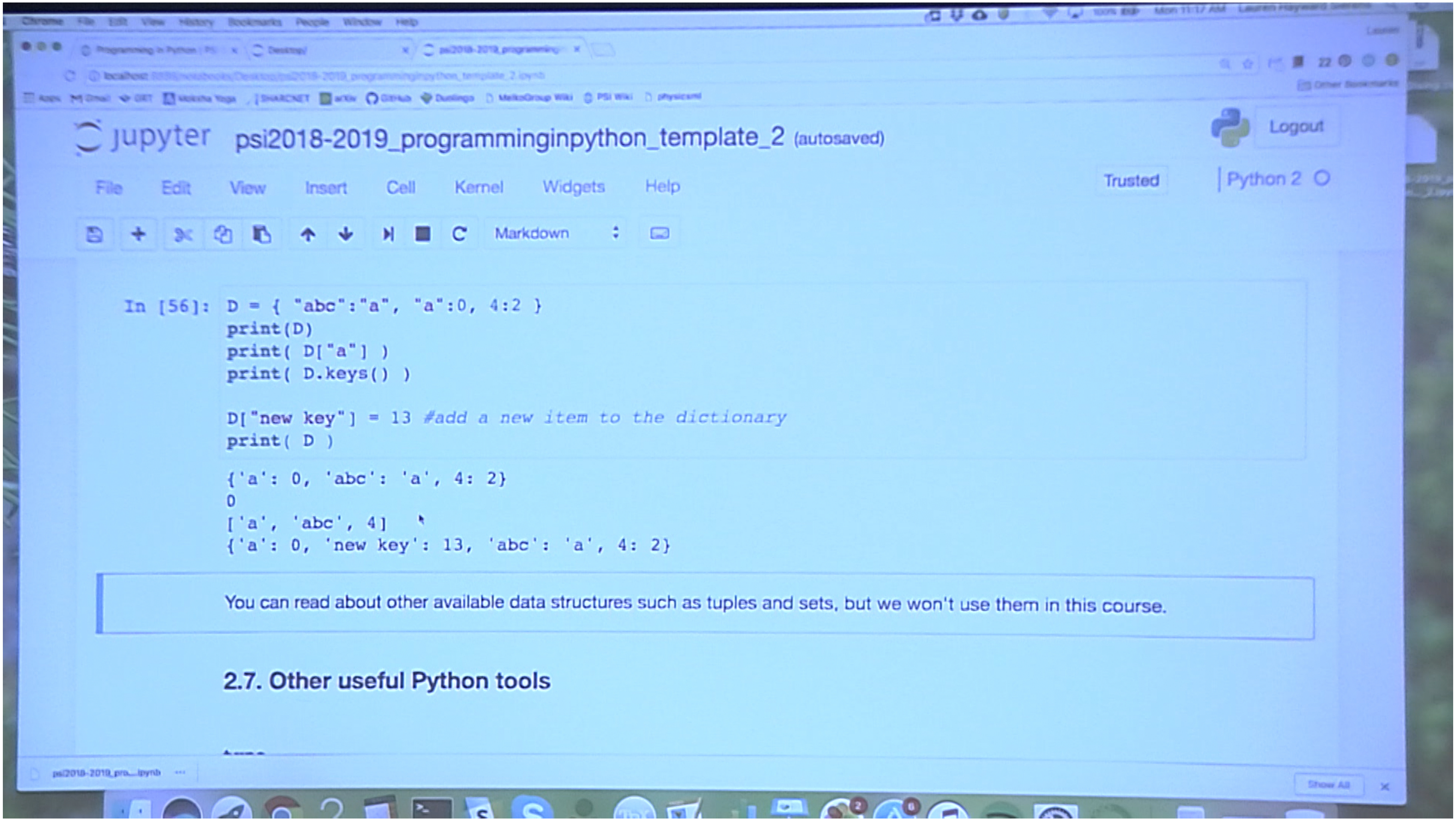
indexed by keys that you define. Each key can be a string, a number, or any other immutable type. You can think of a dictionary as a mapping.

In [56]:
```
D = { "abc":"a", "a":0, 4:2 }
print(D)
print( D["a"] )
print( D.keys() )

D["new key"] = 13 #add a new item to the dictionary
print( D )
```

```
{'a': 0, 'abc': 'a', 4: 2}
0
['a', 'abc', 4]
{'a': 0, 'new key': 13, 'abc': 'a', 4: 2}
```

You can read about other available data structures such as tuples and sets, but we won't use them in this course.

2.7. Other useful Python tools


jupyter psi2018-2019_programminginpython_template_2 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
In [56]: D = { "abc":"a", "a":0, 4:2 }
print(D)
print(D["a"])
print(D.keys())
D["new key"] = 13 #add a new item to the dictionary
print(D)
{'a': 0, 'abc': 'a', 4: 2}
0
['a', 'abc', 4]
{'a': 0, 'new key': 13, 'abc': 'a', 4: 2}
You can read about other available data structures such as tuples and sets, but we won't use them in this course.
2.7. Other useful Python tools

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

You can read about other available data structures such as tuples and sets, but we won't use them in this course.

2.7. Other useful Python tools

`type`

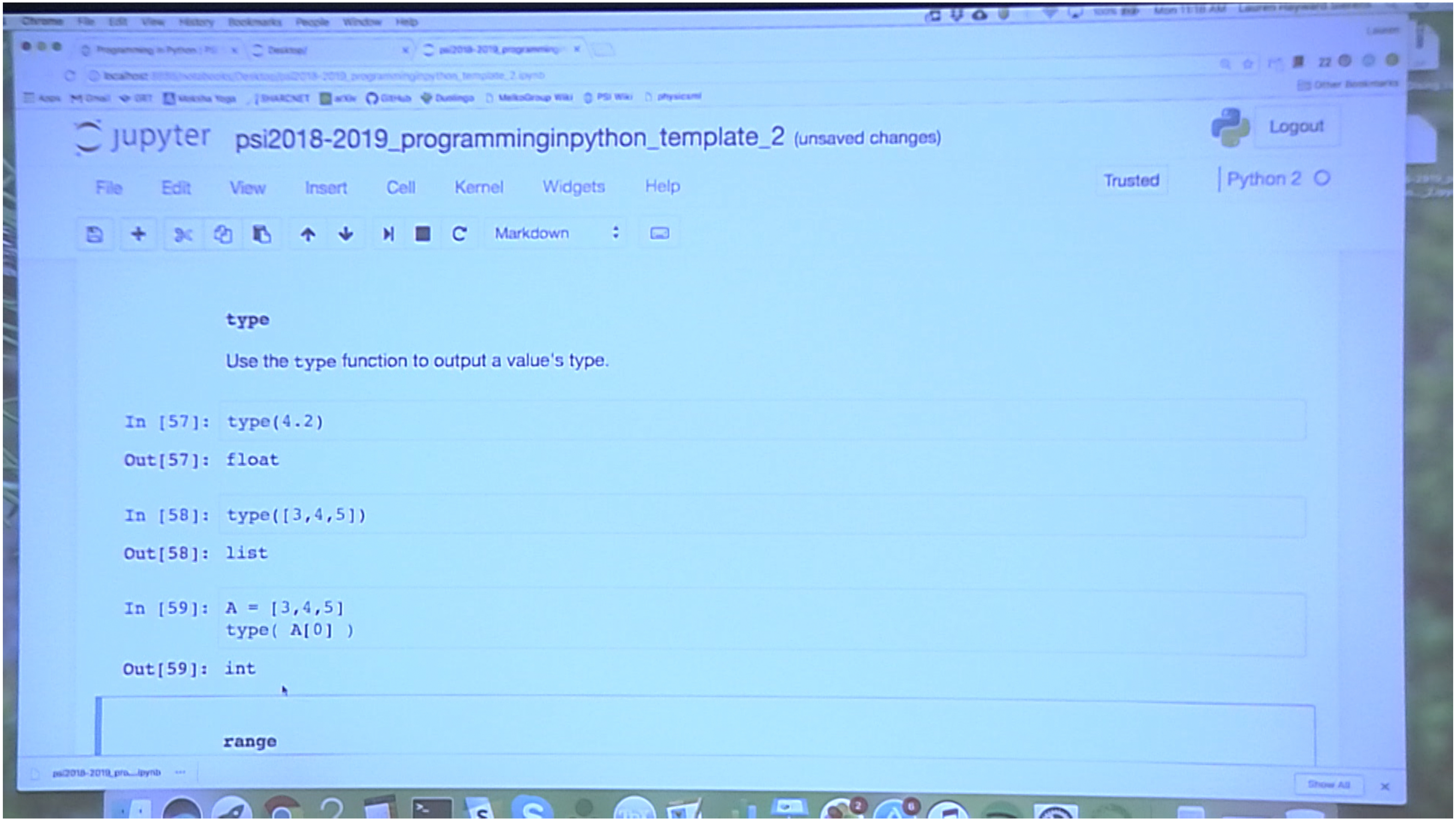
Use the `type` function to output a value's type.

```
In [ ]: 
```

`range`

The `range` function can be used to generate lists that contain sequences of numbers.

```
In [12]: #With one argument, range(stop) gives a sequence of integers from 0 to stop-1:
```


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
type
Use the type function to output a value's type.
In [57]: type(4.2)
Out[57]: float
In [58]: type([3,4,5])
Out[58]: list
In [59]: A = [3,4,5]
type(A[0])
Out[59]: int
range

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

```
In [58]: type([3,4,5])

Out[58]: list

In [59]: A = [3,4,5]
         type( A[0] )

Out[59]: int
```

range

The `range` function can be used to generate lists that contain sequences of numbers.

```
In [12]: #With one argument, range(stop) gives a sequence of integers from 0 to stop-1:
         print( range(12) )

         #With two arguments, range(start,stop) gives a sequence of integers from start to stop-1

         #An optional third argument is the step size:
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

Markdown

```
#An optional third argument is the step size:
print( range(2, 12, 5) )
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[2, 7]
```

`import` and `time`

Often, we will want to use a package or module that is not included by default. In this case we can use the `import` command to import packages, functions, modules or submodules.

One such useful module is called `time` and can be used for timing how long code takes to run.

In [13]:
```
import time #import the time module
```

3. Control structures

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

```
#An optional third argument is the step size:
print( range(2, 12, 5) )
```

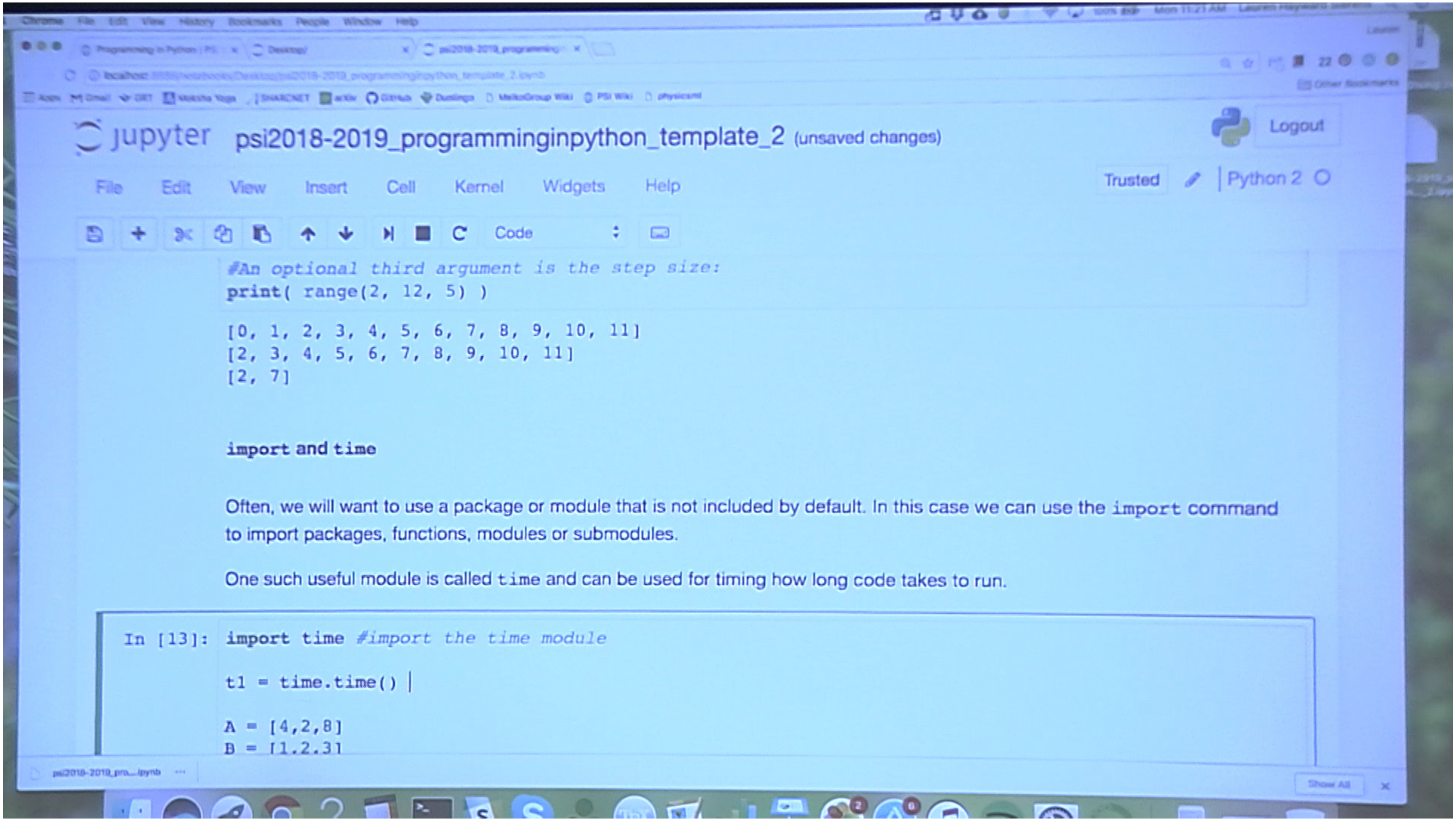
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[2, 7]
```

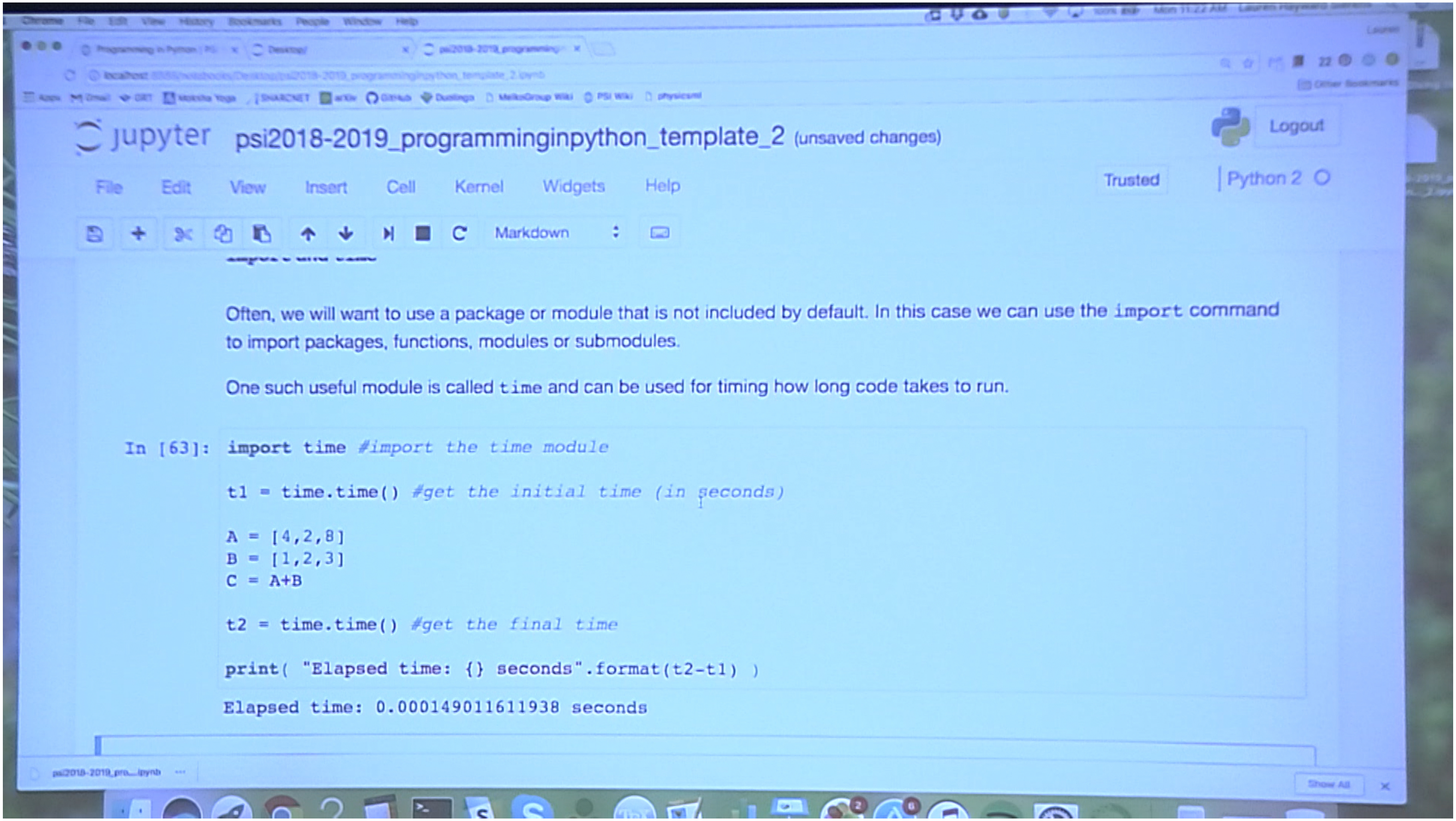
`import` and `time`

Often, we will want to use a package or module that is not included by default. In this case we can use the `import` command to import packages, functions, modules or submodules.

One such useful module is called `time` and can be used for timing how long code takes to run.

In [13]:
```
import time #import the time module

A = [4,2,8]
B = [1,2,3]
C = A+B
```


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
#An optional third argument is the step size:
print(range(2, 12, 5))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[2, 7]
import and time
Often, we will want to use a package or module that is not included by default. In this case we can use the import command to import packages, functions, modules or submodules.
One such useful module is called time and can be used for timing how long code takes to run.
In [13]: import time #import the time module
t1 = time.time()
A = [4,2,8]
B = [1,2,3]


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 2
Markdown
Often, we will want to use a package or module that is not included by default. In this case we can use the import command to import packages, functions, modules or submodules.
One such useful module is called time and can be used for timing how long code takes to run.
In [63]:
import time #import the time module
t1 = time.time() #get the initial time (in seconds)
A = [4,2,8]
B = [1,2,3]
C = A+B
t2 = time.time() #get the final time
print("Elapsed time: {} seconds".format(t2-t1))
Elapsed time: 0.000149011611938 seconds

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 2

Markdown

In [63]:
```
import time #import the time module

t1 = time.time() #get the initial time (in seconds)

A = [4,2,8]
B = [1,2,3]
C = A+B

t2 = time.time() #get the final time

print( "Elapsed time: {} seconds".format(t2-t1) )
```

```
Elapsed time: 0.000149011611938 seconds
```

In [64]:
```
%%time

A = [4,2,8]
B = [1,2,3]
C = A+B
```

```
CPU times: user 8 µs, sys: 6 µs, total: 14 µs
```

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

Markdown

3. Control structures

3.0. `if` statement

An `if` statement is used if we want to execute some section of code only if some conditional expression is true.

For example, the code segment

```
if x:
    print("Hello")
```

will print `Hello` only if the *truth value* of `x` is true.

The truth value of `x` is false if it is has the value `None`, `False`, zero (of any numeric type), or if it is empty (or example, if it is an empty list, string or dictionary). Otherwise, the truth value of `x` is true.

```
In [ ]:
```

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

3.6. if statement

An `if` statement is used if we want to execute some section of code only if some conditional expression is true.

For example, the code segment

```
if x:
    print("Hello")
```

will print `Hello` only if the *truth value* of `x` is true.

The truth value of `x` is false if it is has the value `None`, `False`, zero (of any numeric type), or if it is empty (or example, if it is an empty list, string or dictionary). Otherwise, the truth value of `x` is true.

```
In [ ]:
```

Examples of conditional expressions are:

Conditional expressions	Description
`a or b`	logical OR

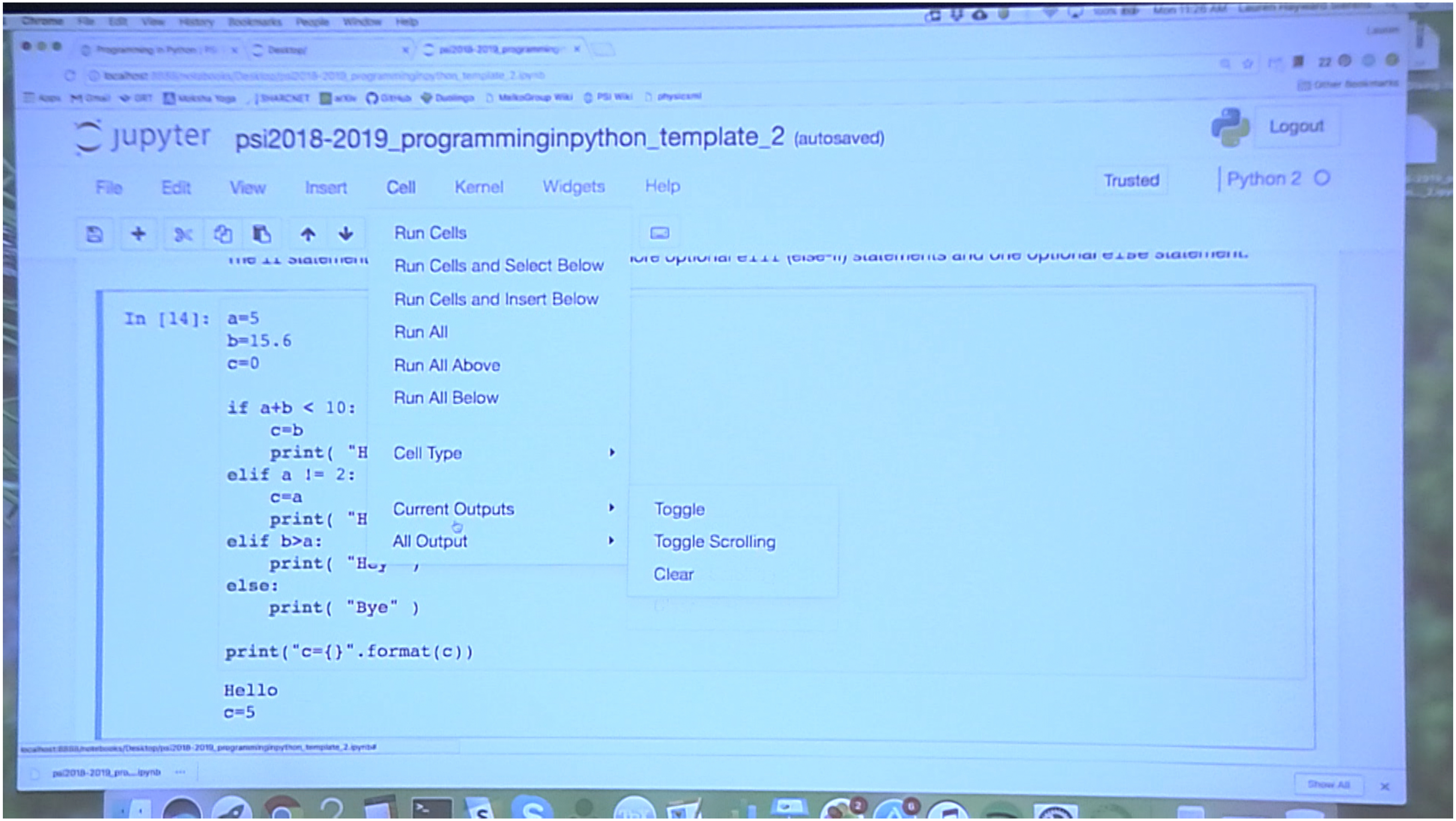
jupyter psi2018-2019_programminginpython_template_2 (autosaved)

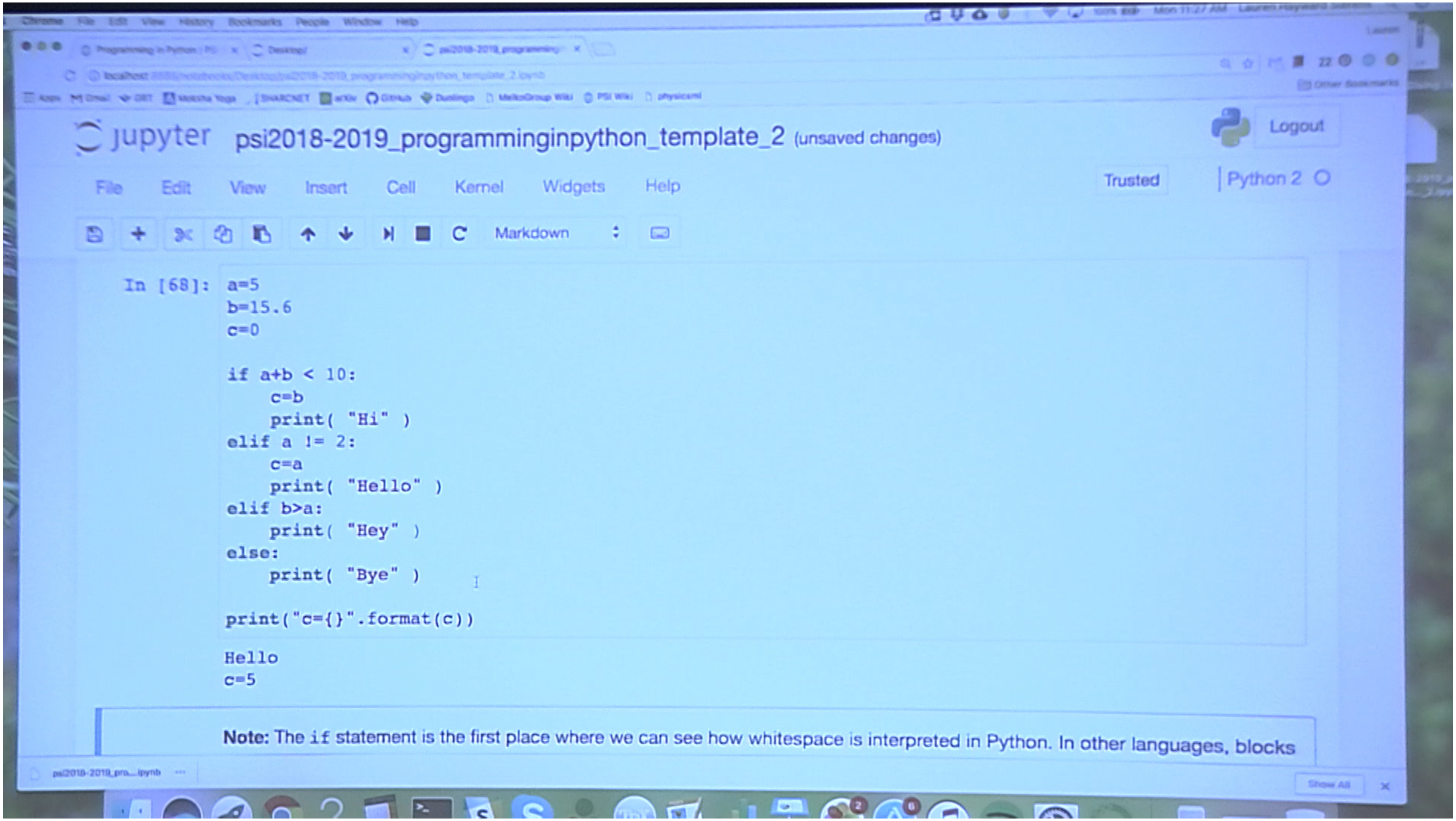
```
Hello
```

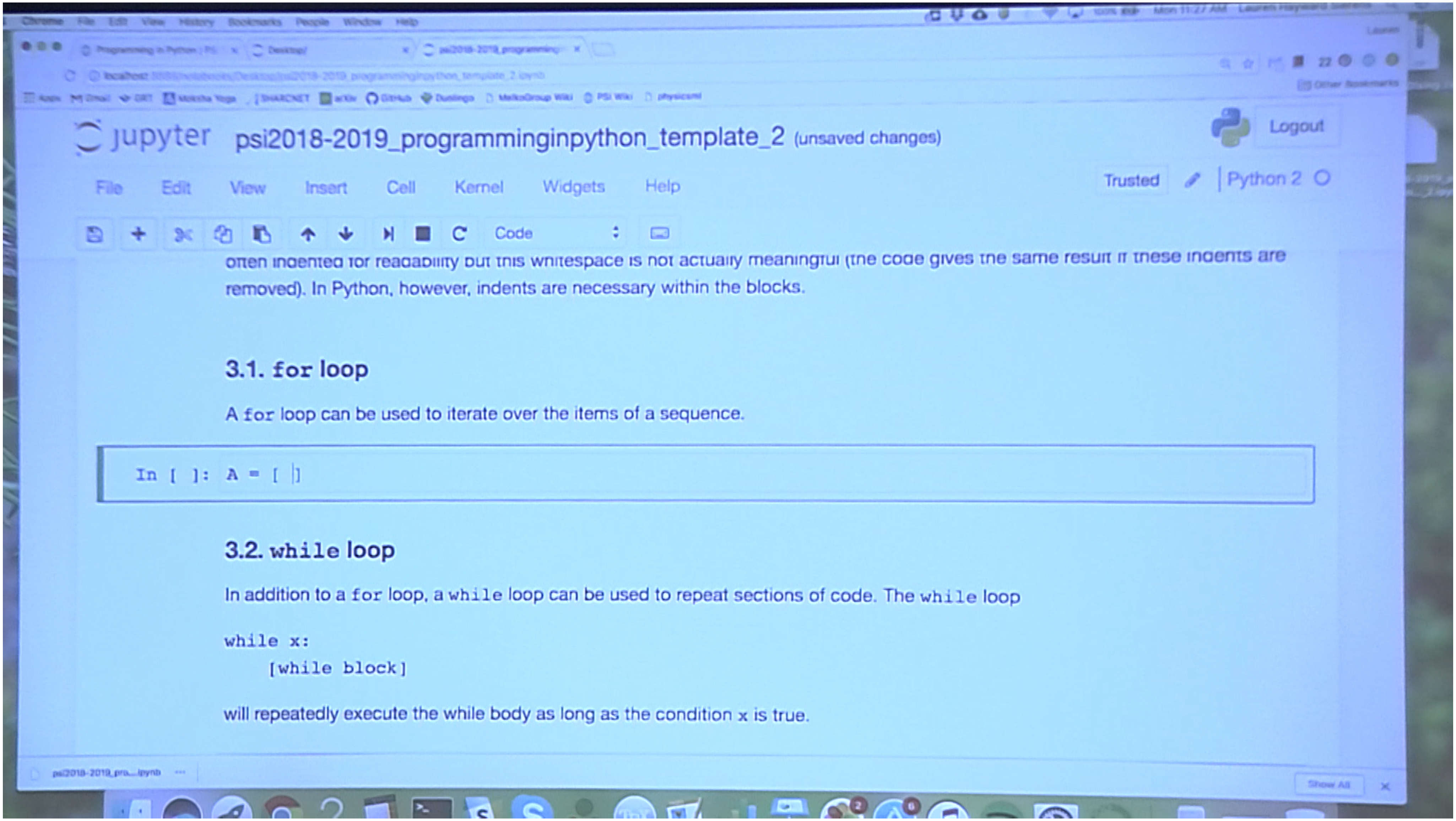
Examples of conditional expressions are:

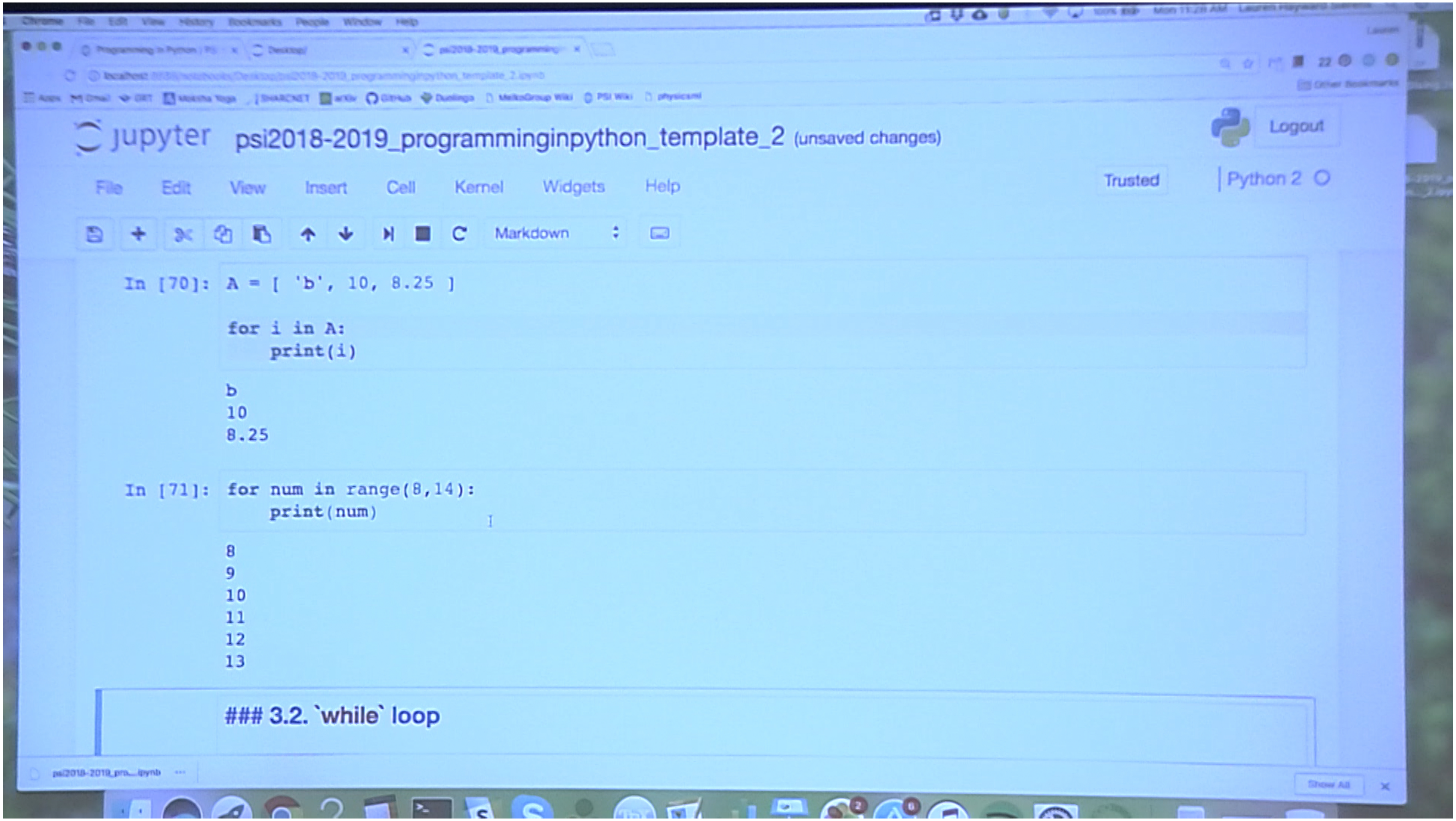
Conditional expressions	Description
`a or b`	logical OR
`a and b`	logical AND
`not a`	logical NOT
`a < b`	less than
`a <= b`	less than or equal to
`a > b`	greater than
`a >= b`	greater than or equal to
`a == b`	equal

The `if` statement can be combined with one or more optional `elif` (else-if) statements and one optional `else` statement.


jupyter psi2018-2019_programminginpython_template_2 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Run Cells
Run Cells and Select Below
Run Cells and Insert Below
Run All
Run All Above
Run All Below
Cell Type
Current Outputs
All Output
Toggle
Toggle Scrolling
Clear
In [14]: a=5
b=15.6
c=0
if a+b < 10:
c=b
print("H
elif a != 2:
c=a
print("H
elif b>a:
else:
print("Bye")
print("c={}".format(c))
Hello
c=5


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
In [68]:
a=5
b=15.6
c=0
if a+b < 10:
c=b
print("Hi")
elif a != 2:
c=a
print("Hello")
elif b>a:
print("Hey")
else:
print("Bye")
print("c={}".format(c))
Hello
c=5
Note: The if statement is the first place where we can see how whitespace is interpreted in Python. In other languages, blocks


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
often indented for readability but this whitespace is not actually meaningful (the code gives the same result if these indents are removed). In Python, however, indents are necessary within the blocks.
3.1. for loop
A for loop can be used to iterate over the items of a sequence.
In [ ]: A = [ ]
3.2. while loop
In addition to a for loop, a while loop can be used to repeat sections of code. The while loop
while x:
[while block]
will repeatedly execute the while body as long as the condition x is true.


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
In [70]: A = ['b', 10, 8.25]
for i in A:
    print(i)
b
10
8.25
In [71]: for num in range(8,14):
    print(num)
8
9
10
11
12
13
3.2. `while` loop

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

In addition to a `for` loop, a `while` loop can be used to repeat sections of code. The `while` loop

```
while x:
    [while block]
```

will repeatedly execute the while body as long as the condition `x` is true.

In [72]:
```
a = 10

while a>0:
    print(a)
    a = a-2
```

```
10
8
6
4
2
```

3.3 Defining functions

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

```
8
6
4
2
```

3.3. Defining functions

Use the keyword `def` to define your own function. This function can then be called from other parts of your code. Optionally, your function can accept one or more input parameters. As another option, you can use a `return` statement to return one or more values from your function.

```
In [15]: #Function with no input parameters:
```

```
In [16]: #Function with two input parameters:
```

```
In [17]: #Function with one input parameter and two return values:
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

3.3. Defining functions

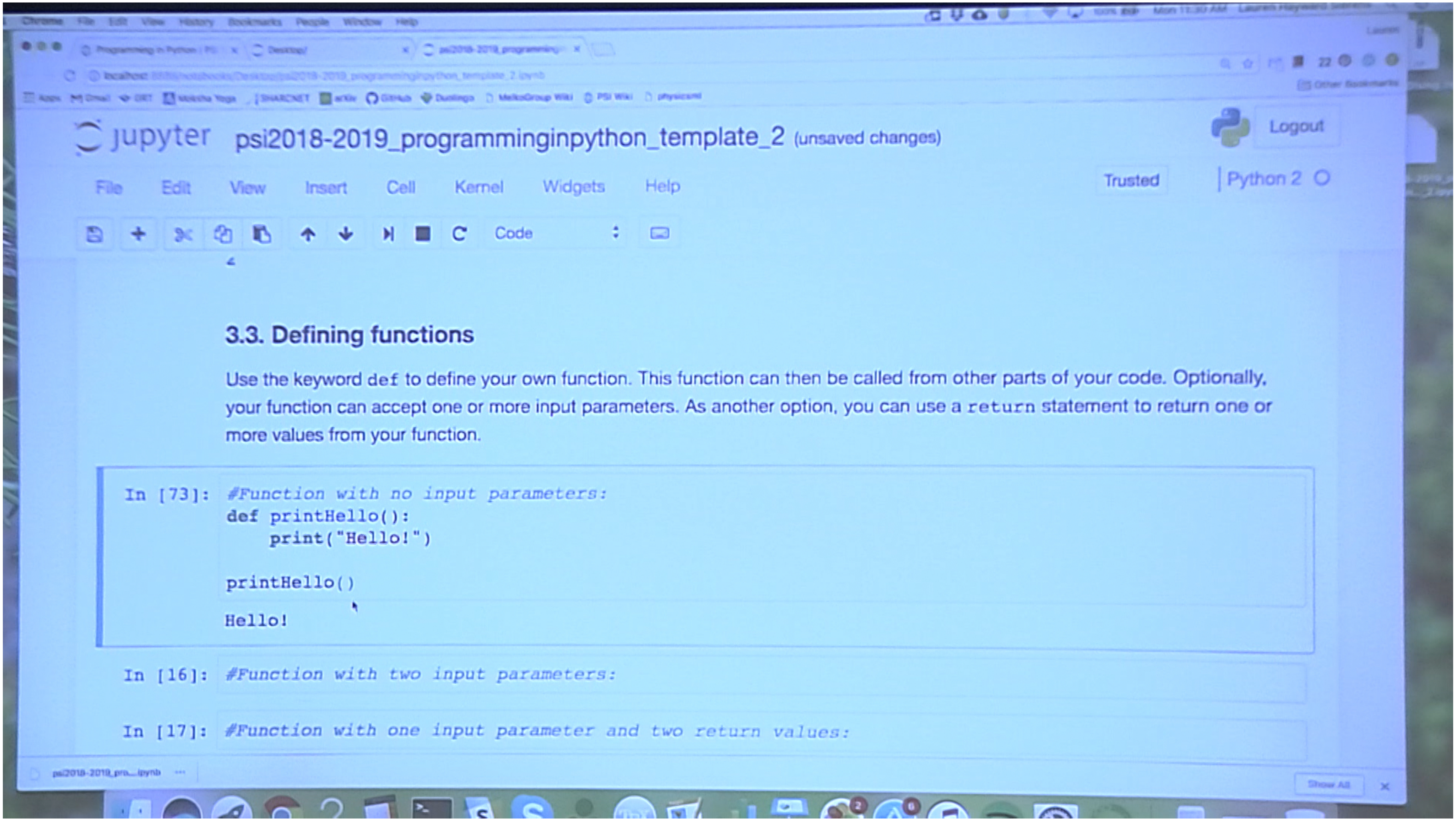
Use the keyword `def` to define your own function. This function can then be called from other parts of your code. Optionally, your function can accept one or more input parameters. As another option, you can use a `return` statement to return one or more values from your function.

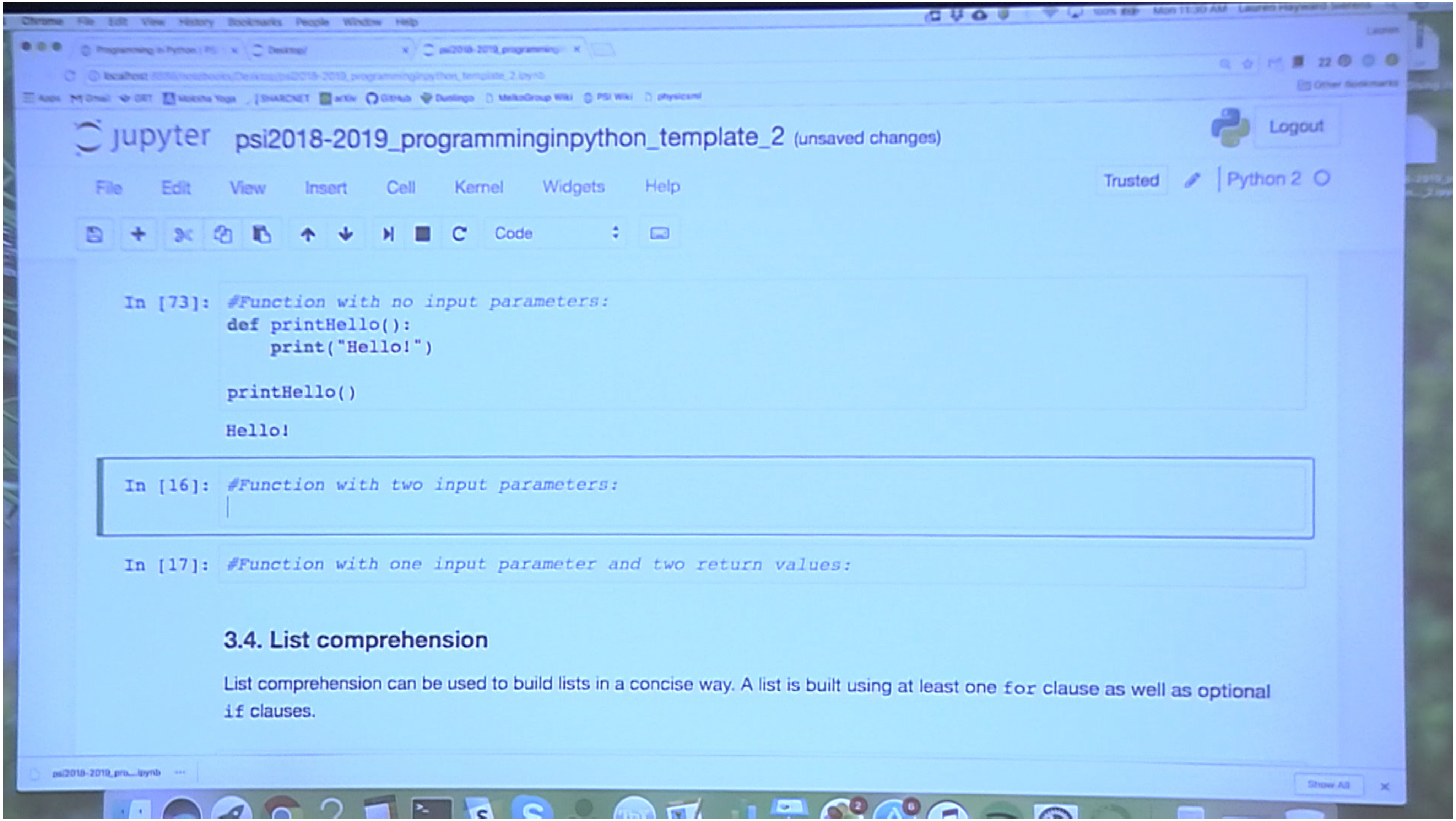
In [73]:
```
#Function with no input parameters:
def printHello():
    print("Hello!")

printHello()
```

```
Hello!
```

In [16]:
```
#Function with two input parameters:
```

In [17]:
```
#Function with one input parameter and two return values:
```


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 2
Code
3.3. Defining functions
Use the keyword def to define your own function. This function can then be called from other parts of your code. Optionally, your function can accept one or more input parameters. As another option, you can use a return statement to return one or more values from your function.
In [73]: #Function with no input parameters:
def printHello():
    print("Hello!")
printHello()
Hello!
In [16]: #Function with two input parameters:
In [17]: #Function with one input parameter and two return values:


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 2
Code
In [73]:
#Function with no input parameters:
def printHello():
print("Hello!")
printHello()
Hello!
In [16]:
#Function with two input parameters:
In [17]:
#Function with one input parameter and two return values:
3.4. List comprehension
List comprehension can be used to build lists in a concise way. A list is built using at least one for clause as well as optional if clauses.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

Code

In [73]:
```
#Function with no input parameters:
def printHello():
    print("Hello!")

printHello()
```

```
Hello!
```

In [16]:
```
#Function with two input parameters:
def get_c(a,b):
    c = 5*a - b
```

In [17]:
```
#Function with one input parameter and two return values:
```

3.4. List comprehension

List comprehension can be used to build lists in a concise way. A list is built using at least one `for` clause as well as optional `if` clauses

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

In [74]:
```
#Function with two input parameters:
def get_c(a,b):
    c = 5*a - b
    print(c)

get_c(1,2)
get_c(4.3,-2)
```

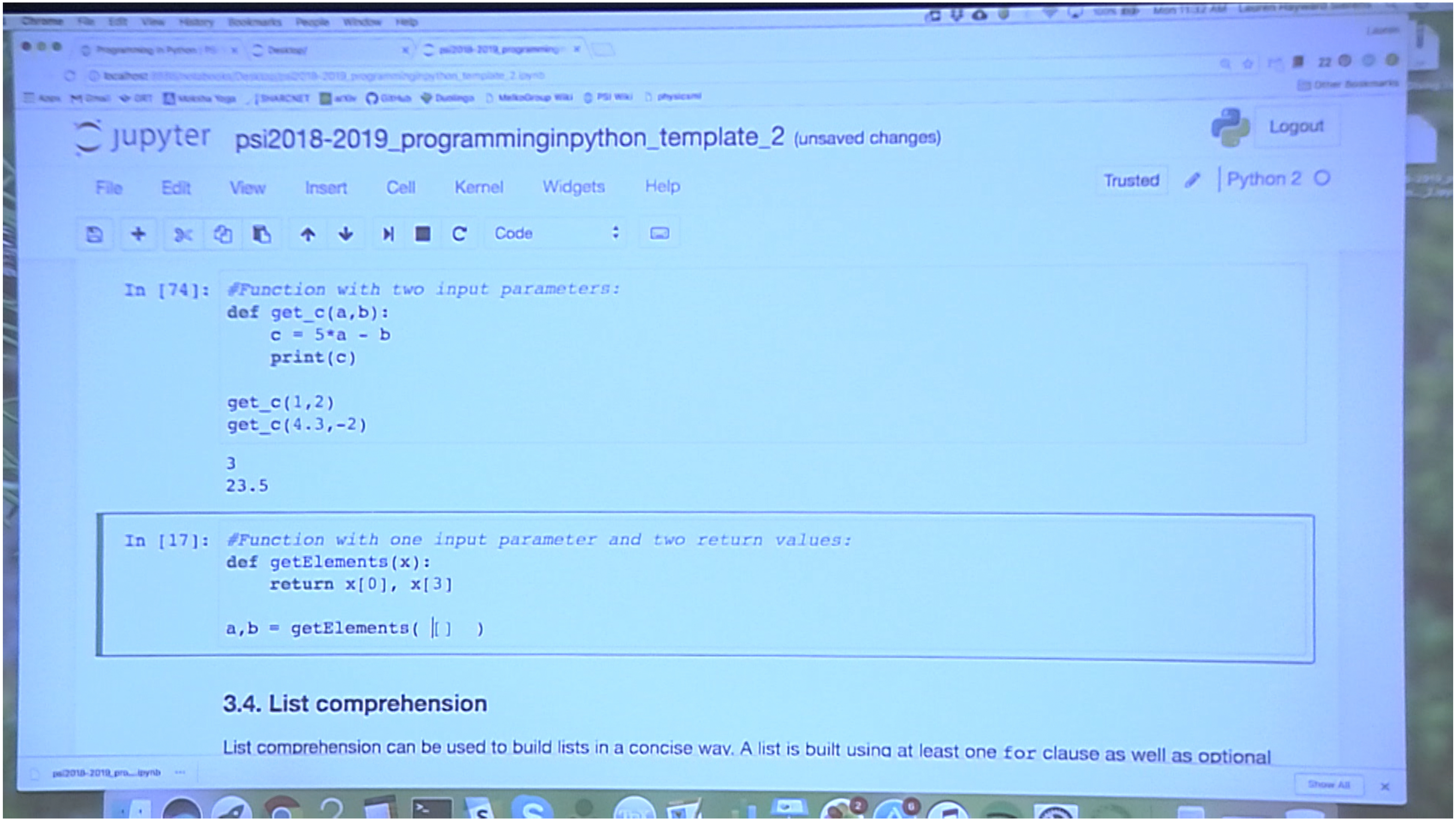
```
3
23.5
```

In [17]:
```
#Function with one input parameter and two return values:
```

3.4. List comprehension

List comprehension can be used to build lists in a concise way. A list is built using at least one `for` clause as well as optional `if` clauses.

In [ ]:


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 2
Code
In [74]:
#Function with two input parameters:
def get_c(a,b):
c = 5*a - b
print(c)
get_c(1,2)
get_c(4.3,-2)
3
23.5
In [17]:
#Function with one input parameter and two return values:
def getElements(x):
return x[0], x[3]
a,b = getElements([])
3.4. List comprehension
List comprehension can be used to build lists in a concise way. A list is built using at least one for clause as well as optional

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 2

Code

```
def get_c(a,b):
    c = 5*a - b
    print(c)

get_c(1,2)
get_c(4.3,-2)
```

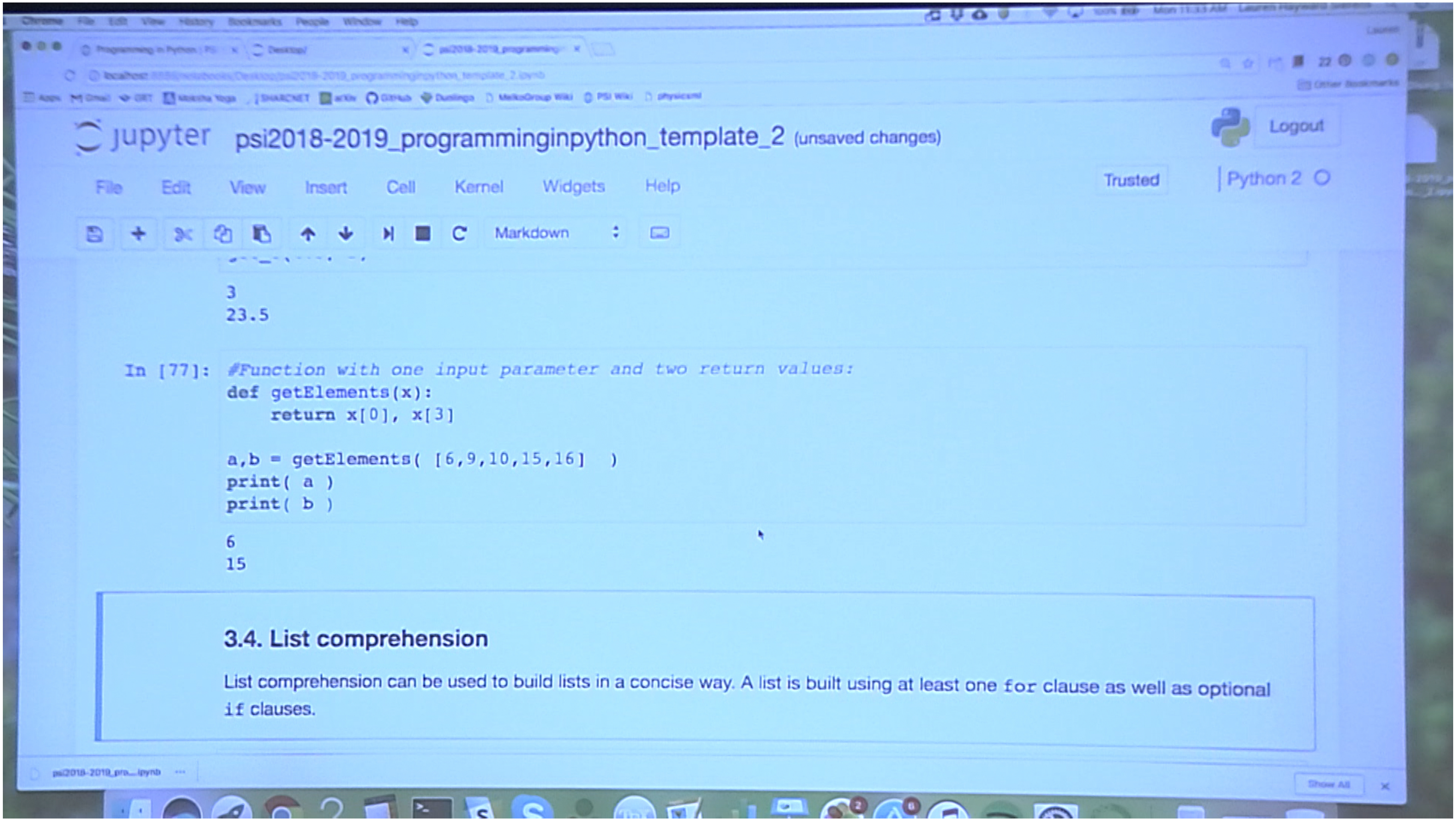
```
3
23.5
```

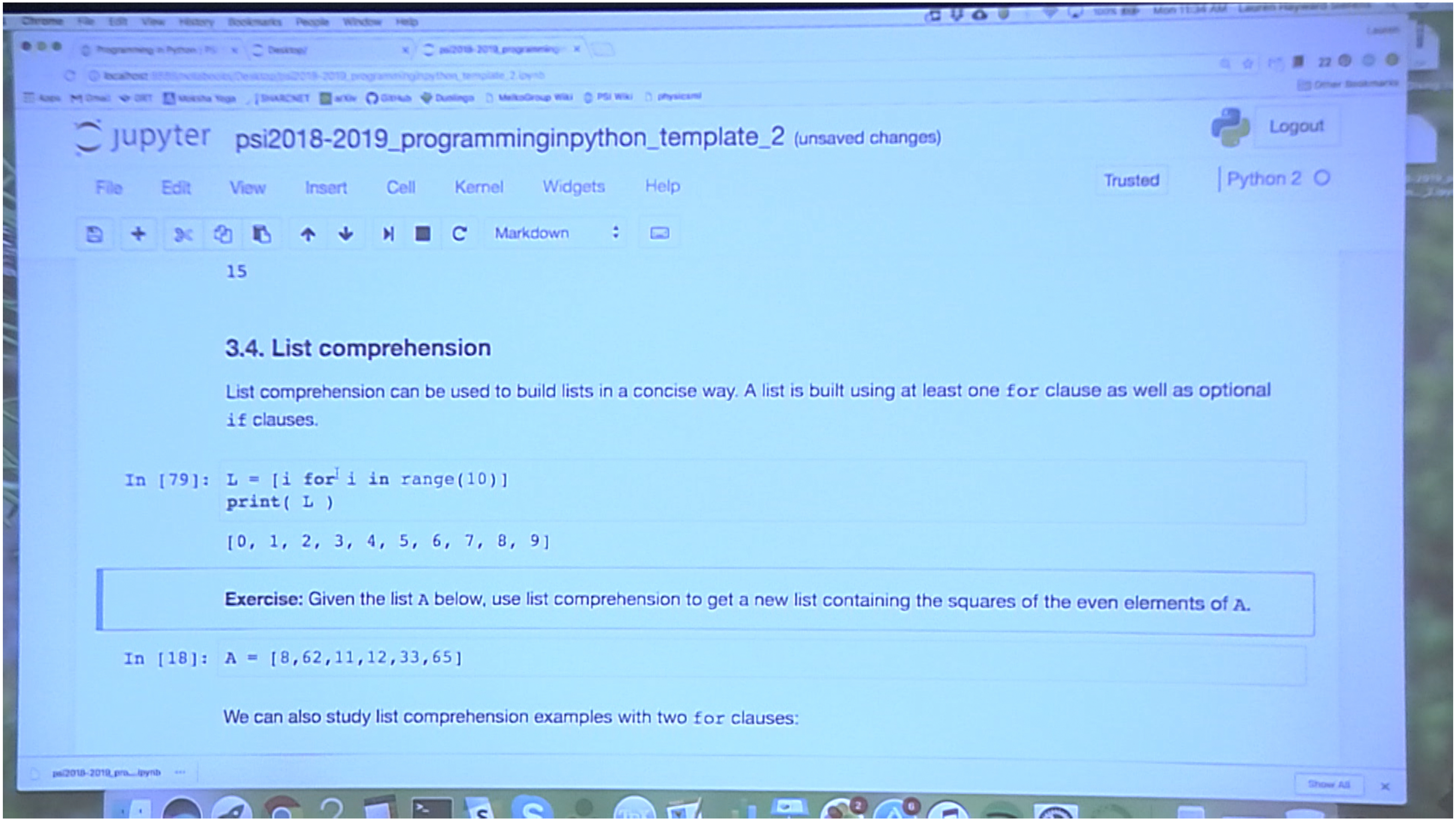
In [76]:

```
#Function with one input parameter and two return values:
def getElements(x):
    return x[0], x[3]

a,b = getElements( [6,9,10,15,16]  )
```

3.4. List comprehension

List comprehension can be used to build lists in a concise way. A list is built using at least one `for` clause as well as optional `if` clauses.


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 2
Markdown
3
23.5
In [77]:
#Function with one input parameter and two return values:
def getElements(x):
    return x[0], x[3]
a,b = getElements([6,9,10,15,16])
print(a)
print(b)
6
15
3.4. List comprehension
List comprehension can be used to build lists in a concise way. A list is built using at least one for clause as well as optional if clauses.


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
15
3.4. List comprehension
List comprehension can be used to build lists in a concise way. A list is built using at least one for clause as well as optional if clauses.
In [79]: L = [i for i in range(10)]
print(L)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Exercise: Given the list A below, use list comprehension to get a new list containing the squares of the even elements of A.
In [18]: A = [8,62,11,12,33,65]
We can also study list comprehension examples with two for clauses:

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

```
15
```

3.4. List comprehension

List comprehension can be used to build lists in a concise way. A list is built using at least one `for` clause as well as optional `if` clauses.

In [79]:
```
L = [i for i in range(10)]
print( L )
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [ ]:
```
A = [7,-8,2,20]
B = [a*2 for a ]
```

Exercise: Given the list `A` below, use list comprehension to get a new list containing the squares of the even elements of `A`.

In [18]:
```
A = [8,62,11,12,33,65]
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

List comprehension can be used to build lists in a concise way. A list is built using at least one `for` clause as well as optional `if` clauses.

In [79]:
```
L = [i for i in range(10)]
print( L )
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [81]:
```
A = [7,-8,2,20]
B = [a*2 for a in A if a>0]
print( B )
```

```
[14, 4, 40]
```

Exercise: Given the list `A` below, use list comprehension to get a new list containing the squares of the even elements of `A`.

In [18]:
```
A = [8,62,11,12,33,65]
```

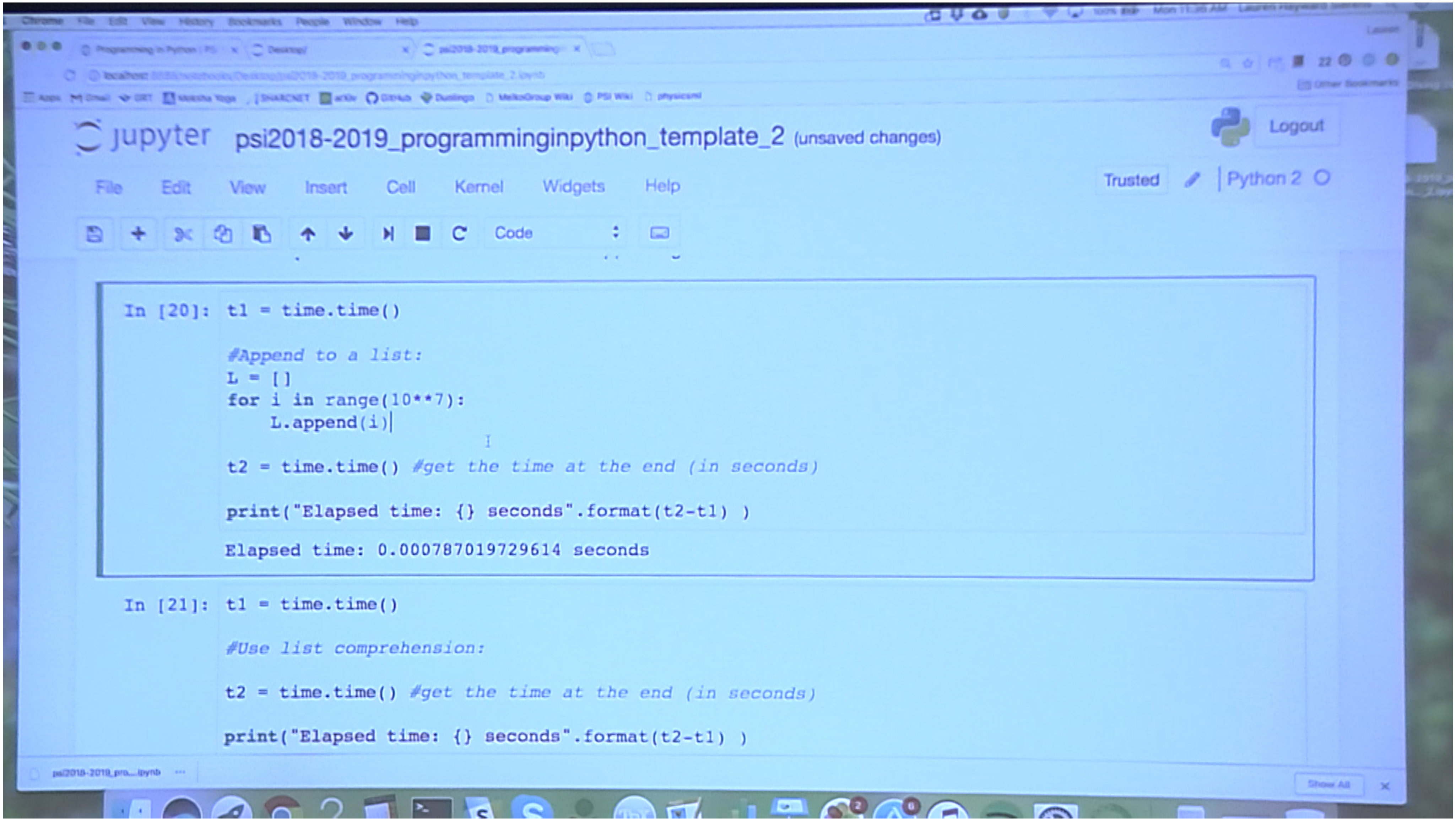
We can also study list comprehension examples with two `for` clauses:

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

```
In [19]: A = [3,4]
         B = [4,7,11]
```

List comprehension can be much faster than appending to a list.

```
In [20]: t1 = time.time()

         #Append to a list:

         t2 = time.time() #get the time at the end (in seconds)

         print("Elapsed time: {} seconds".format(t2-t1) )

         Elapsed time: 0.000787019729614 seconds

In [21]: t1 = time.time()

         #Use list comprehension:

         t2 = time.time() #get the time at the end (in seconds)
```


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
In [20]:
t1 = time.time()

#Append to a list:
L = []
for i in range(10**7):
    L.append(i)

t2 = time.time() #get the time at the end (in seconds)

print("Elapsed time: {} seconds".format(t2-t1))
Elapsed time: 0.000787019729614 seconds
In [21]:
t1 = time.time()

#Use list comprehension:

t2 = time.time() #get the time at the end (in seconds)

print("Elapsed time: {} seconds".format(t2-t1))

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

```
print("Elapsed time: {} seconds".format(t2-t1) )
```

Elapsed time: 1.53906798363 seconds

In [83]:
```
t1 = time.time()

#Use list comprehension:
L = [i for i in range(10**7)]

t2 = time.time() #get the time at the end (in seconds)

print("Elapsed time: {} seconds".format(t2-t1) )
```

Elapsed time: 1.12796211243 seconds

4. The NumPy package

Here we will explore some of the scientific computing functionality available through the NumPy package.

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Notebook saved Trusted Python 2

```
print("Elapsed time: {} seconds".format(t2-t1) )
```

```
Elapsed time: 1.12796211243 seconds
```

4. The NumPy package

Here we will explore some of the scientific computing functionality available through the NumPy package.

Whenever we want to use NumPy, we must first import it. A standard convention is to import it under the name `np`.

```
In [22]: import numpy as np
```

4.0. Arrays

The most common objects in NumPy are single- or multi-dimensional arrays. Arrays are tables of values that all have the same type. Here we will only consider arrays of numbers.

There are various functions available for creating NumPy arrays.

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Whenever we want to use NumPy, we must first import it. A standard convention is to import it under the name `np`.

In [84]:
```
import numpy as np
```

4.0. Arrays

The most common objects in NumPy are single- or multi-dimensional arrays. Arrays are tables of values that all have the same type. Here we will only consider arrays of numbers.

There are various functions available for creating NumPy arrays.

In [23]:
```
#Create arrays from other data structures such as lists:
```

In [24]:
```
#Use np.zeros to create arrays of zeros of different sizes and dimensions.
#Similarly, use np.ones to create arrays of ones.
```

In [25]:
```
#Use np.arange to create arrays containing sequences of numbers (similar to range for lists)
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

Code

type. Here we will only consider arrays of numbers.

There are various functions available for creating NumPy arrays.

In [88]:
```
#Create arrays from other data structures such as lists:
A = np.array( [5,-2,13] ) #1D array
print( A )
print

B = np.array( [[5,-2,13], [2,2.2, 3.6]] ) #2D array
print( B )
```

```
[ 5 -2 13]

[[ 5.  -2.  13. ]
 [ 2.   2.2  3.6]]
```

In [24]:
```
#Use np.zeros to create arrays of zeros of different sizes and dimensions.
#Similarly, use np.ones to create arrays of ones.
```

In [25]:
```
#Use np.arange to create arrays containing sequences of numbers (similar to range for lists)
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

```
[ 2.   2.2  3.6]]
```

In [89]:
```
#Use np.zeros to create arrays of zeros of different sizes and dimensions.
#Similarly, use np.ones to create arrays of ones.

np.zeros(3)
```

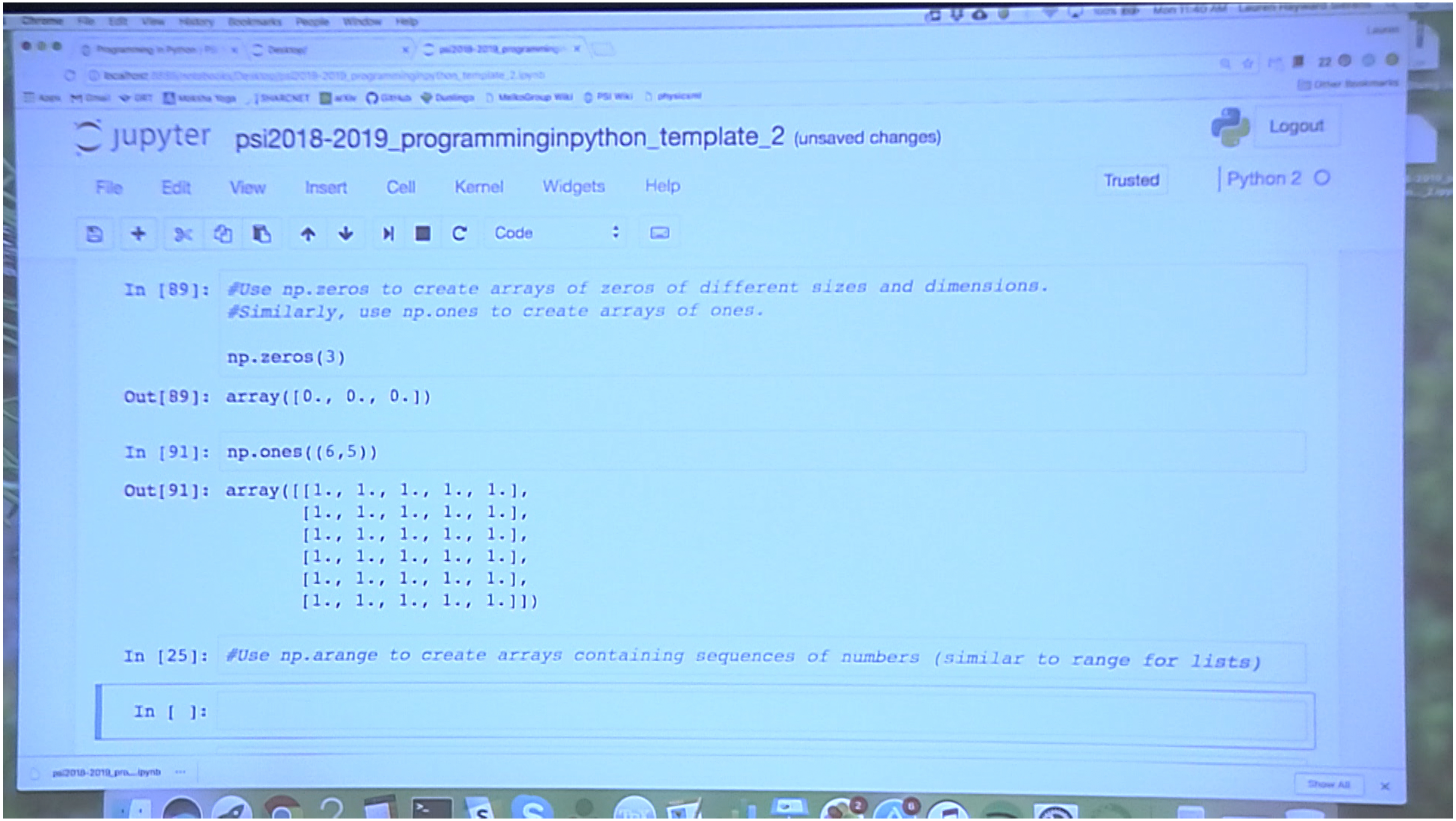
Out[89]:
```
array([0., 0., 0.])
```

In [90]:
```
np.zeros((6,5))
```

Out[90]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [25]:
```
#Use np.arange to create arrays containing sequences of numbers (similar to range for lists)
```

In [26]:
```
#Use np.linspace get an array of linearly spaced numbers between given endpoints
```


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
In [89]: #Use np.zeros to create arrays of zeros of different sizes and dimensions.
#Similarly, use np.ones to create arrays of ones.
np.zeros(3)
Out[89]: array([0., 0., 0.])
In [91]: np.ones((6,5))
Out[91]: array([[1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1.]])
In [25]: #Use np.arange to create arrays containing sequences of numbers (similar to range for lists)
In [ ]:

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

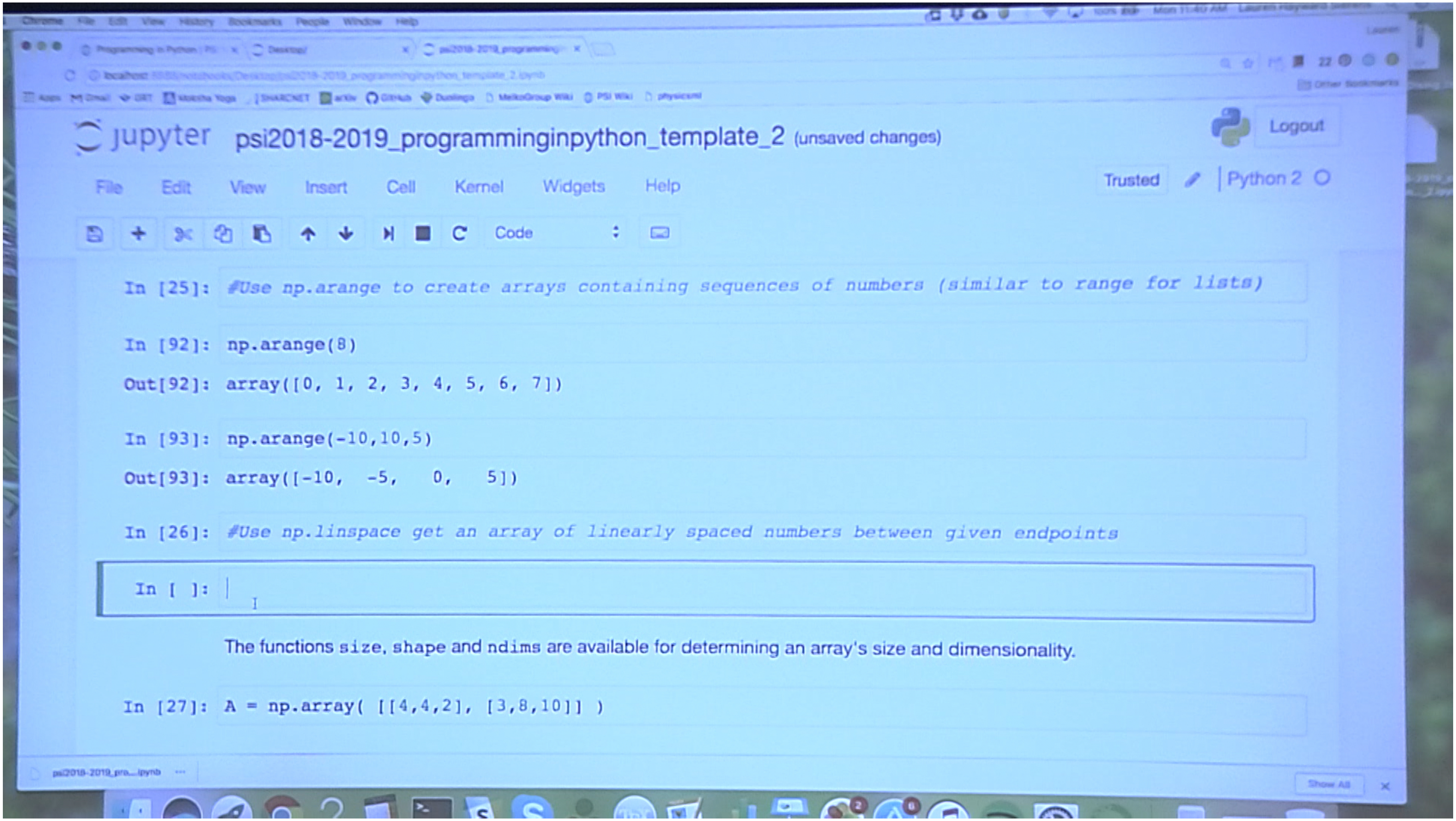
```
Out[91]: array([[1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.]])
```

```
In [25]: #Use np.arange to create arrays containing sequences of numbers (similar to range for lists)
```

```
In [92]: np.arange(8)

Out[92]: array([0, 1, 2, 3, 4, 5, 6, 7])
```

```
In [93]: np.arange(-10,10,5)

Out[93]: array([-10,  -5,   0,   5])
```

```
In [26]: #Use np.linspace get an array of linearly spaced numbers between given endpoints
```

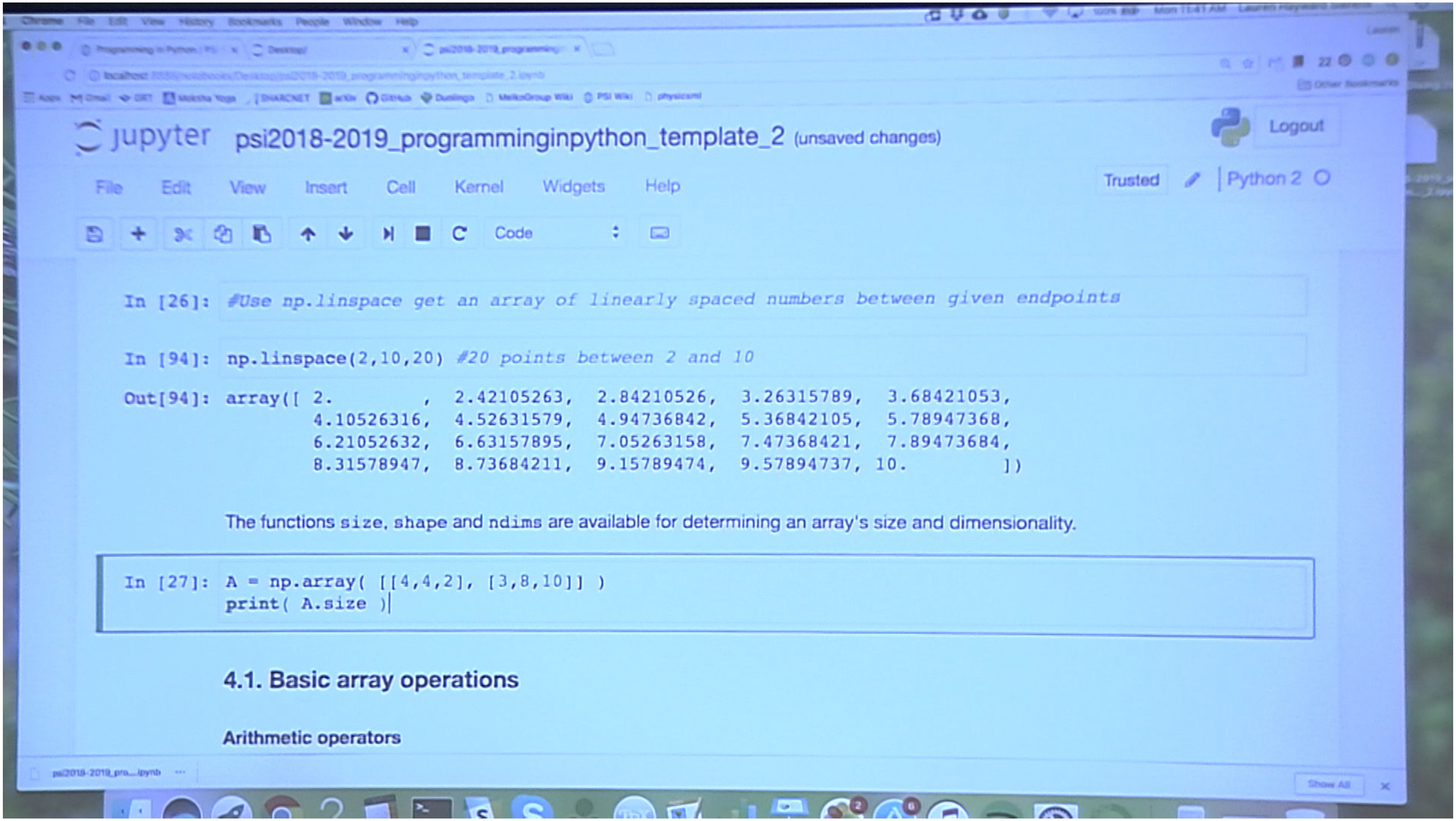

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
In [25]: #Use np.arange to create arrays containing sequences of numbers (similar to range for lists)
In [92]: np.arange(8)
Out[92]: array([0, 1, 2, 3, 4, 5, 6, 7])
In [93]: np.arange(-10,10,5)
Out[93]: array([-10, -5, 0, 5])
In [26]: #Use np.linspace get an array of linearly spaced numbers between given endpoints
In [ ]:
The functions size, shape and ndims are available for determining an array's size and dimensionality.
In [27]: A = np.array([[4,4,2], [3,8,10]])

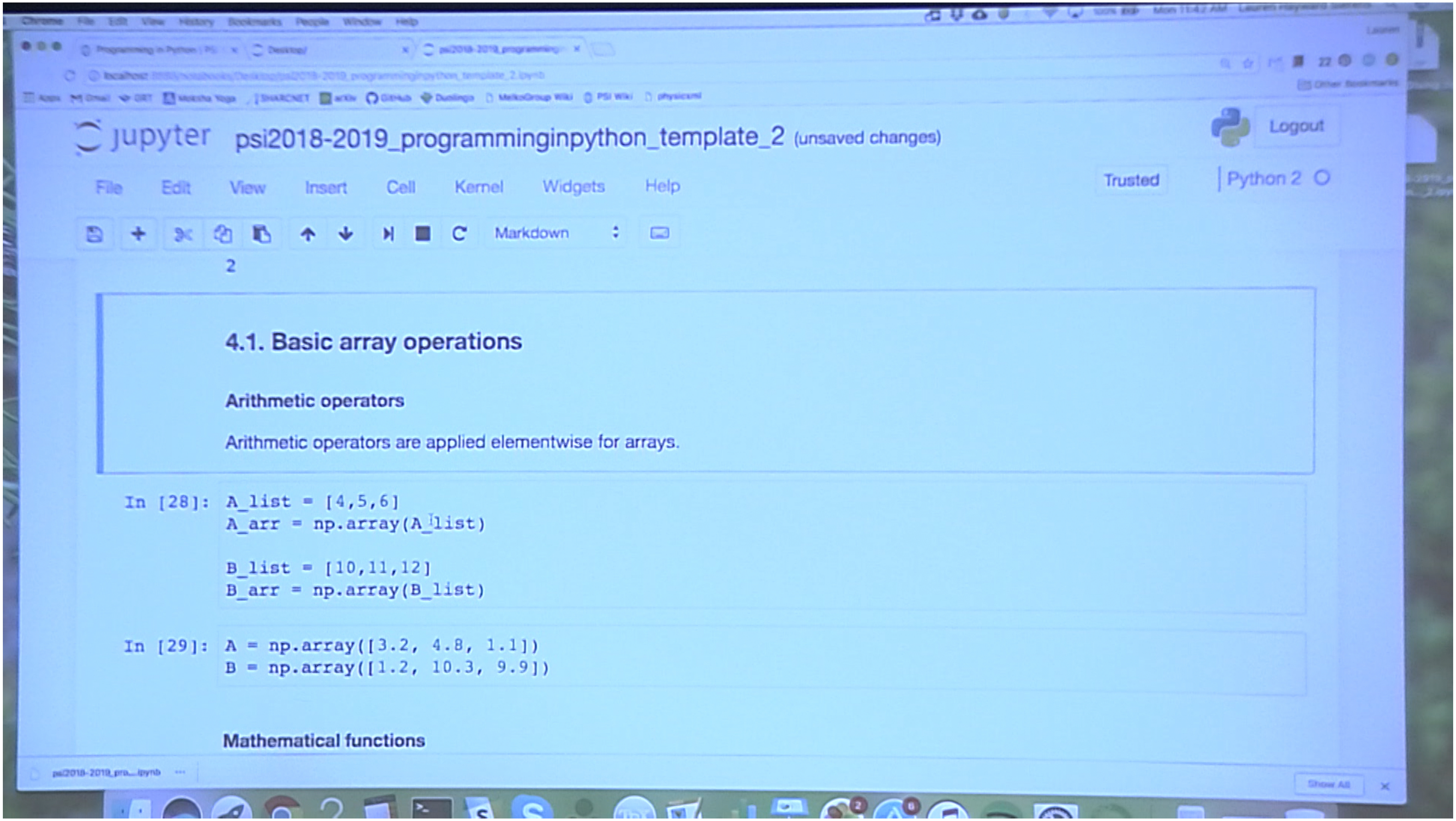
jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

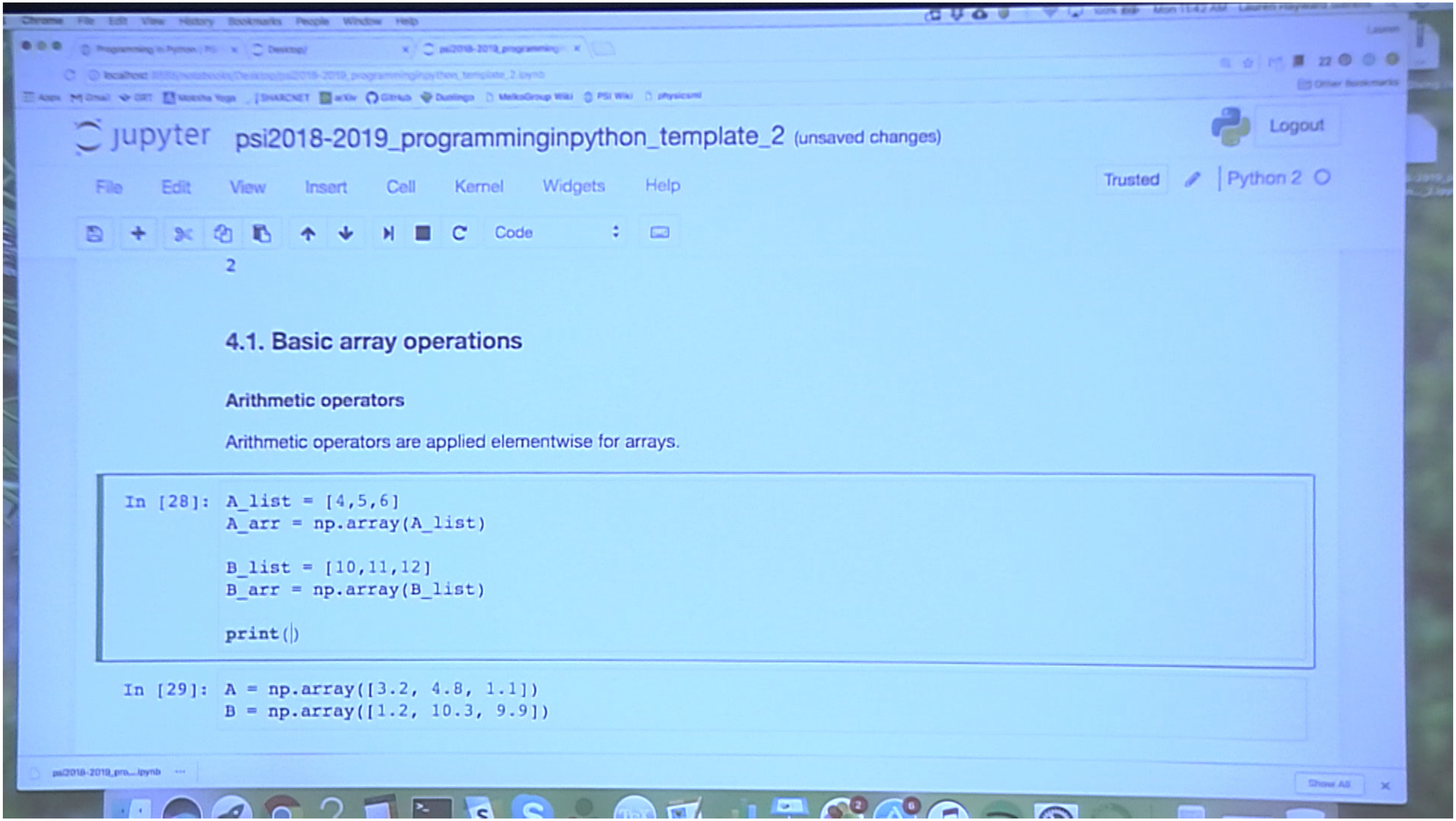
```
Out[92]: array([0, 1, 2, 3, 4, 5, 6, 7])

In [93]: np.arange(-10,10,5)

Out[93]: array([-10,  -5,   0,   5])

In [26]: #Use np.linspace get an array of linearly spaced numbers between given endpoints

In [94]: np.linspace(2,10,20) #20 points between 2 and 10

Out[94]: array([ 2.        ,  2.42105263,  2.84210526,  3.26315789,  3.68421053,
                 4.10526316,  4.52631579,  4.94736842,  5.36842105,  5.78947368,
                 6.21052632,  6.63157895,  7.05263158,  7.47368421,  7.89473684,
                 8.31578947,  8.73684211,  9.15789474,  9.57894737, 10.        ])
```

The functions `size`, `shape` and `ndims` are available for determining an array's size and dimensionality.

```
In [27]: A = np.array( [[4,4,2], [3,8,10]] )
```


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
In [26]: #Use np.linspace get an array of linearly spaced numbers between given endpoints
In [94]: np.linspace(2,10,20) #20 points between 2 and 10
Out[94]: array([2. , 2.42105263, 2.84210526, 3.26315789, 3.68421053,
4.10526316, 4.52631579, 4.94736842, 5.36842105, 5.78947368,
6.21052632, 6.63157895, 7.05263158, 7.47368421, 7.89473684,
8.31578947, 8.73684211, 9.15789474, 9.57894737, 10.])
The functions size, shape and ndims are available for determining an array's size and dimensionality.
In [27]: A = np.array([[4,4,2], [3,8,10]])
print(A.size)
4.1. Basic array operations
Arithmetic operators


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 2
Markdown
2
4.1. Basic array operations
Arithmetic operators
Arithmetic operators are applied elementwise for arrays.
In [28]: A_list = [4,5,6]
A_arr = np.array(A_list)
B_list = [10,11,12]
B_arr = np.array(B_list)
In [29]: A = np.array([3.2, 4.8, 1.1])
B = np.array([1.2, 10.3, 9.9])
Mathematical functions


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 2
Code
2
4.1. Basic array operations
Arithmetic operators
Arithmetic operators are applied elementwise for arrays.
In [28]:
A_list = [4,5,6]
A_arr = np.array(A_list)
B_list = [10,11,12]
B_arr = np.array(B_list)
print()
In [29]:
A = np.array([3.2, 4.8, 1.1])
B = np.array([1.2, 10.3, 9.9])

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

4.1. Basic array operations

Arithmetic operators

Arithmetic operators are applied elementwise for arrays.

In [99]:
```
A_list = [4,5,6]
A_arr = np.array(A_list)

B_list = [10,11,12]
B_arr = np.array(B_list)

print( A_list + B_list )
print( A_arr + B_arr )
```

```
[4, 5, 6, 10, 11, 12]
[14 16 18]
```

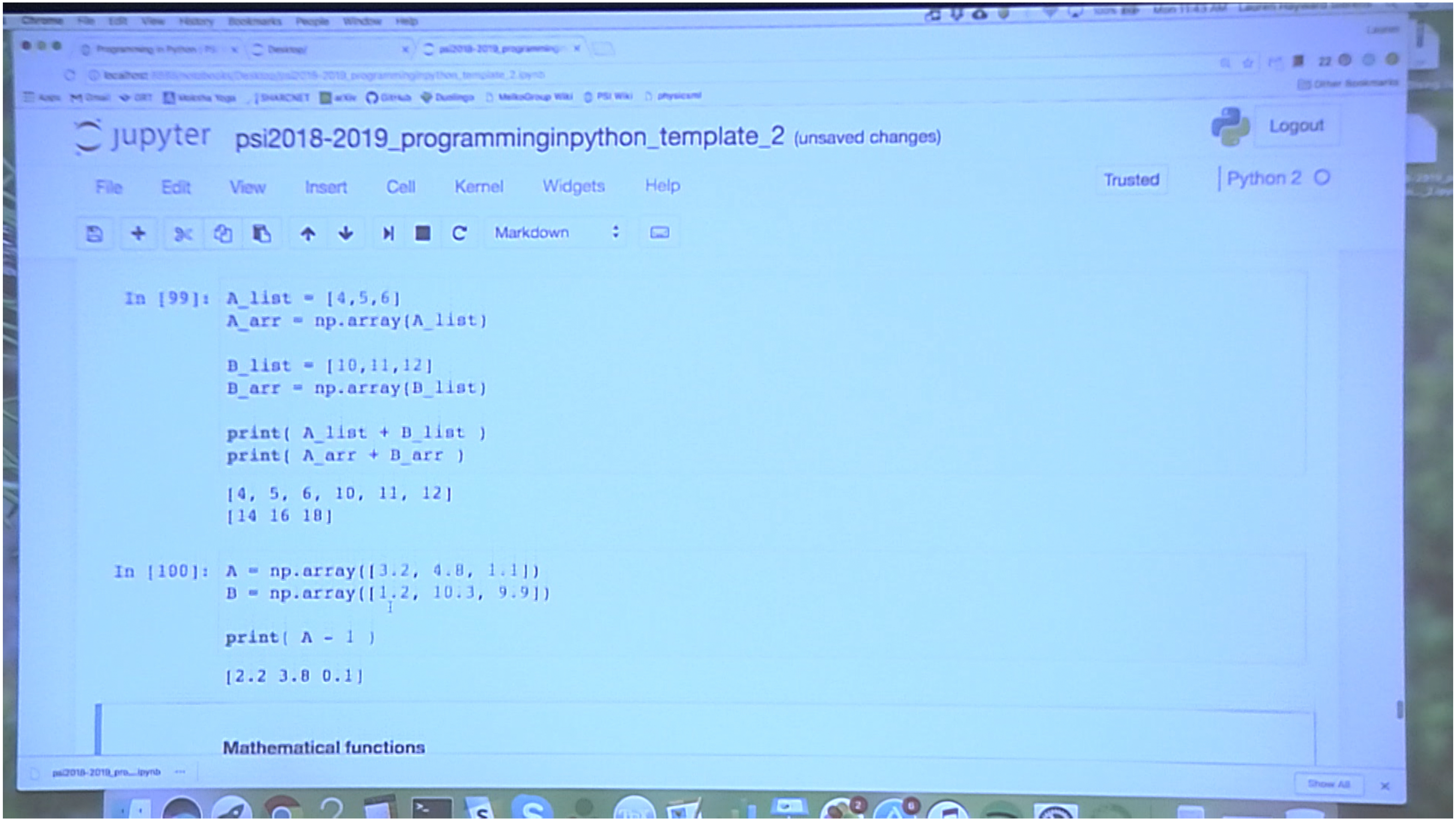
In [29]:
```
A = np.array([3.2, 4.8, 1.1])
B = np.array([1.2, 10.3, 9.9])
```

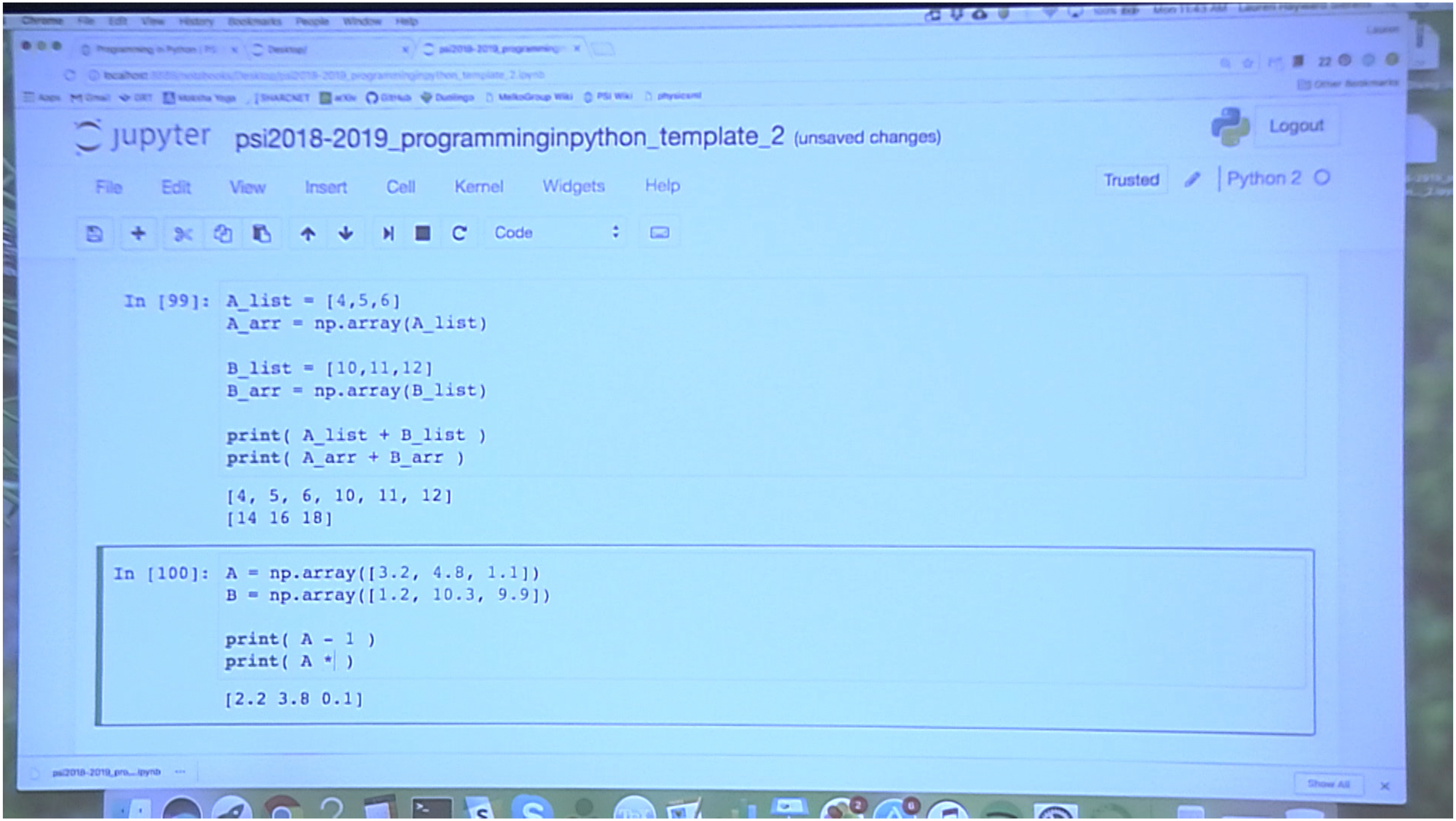
jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Mathematical functions

Numpy has several built-in mathematical functions that act elementwise on arrays `x`, such as:

NumPy function	Description
`np.sqrt(x)`	elementwise square-root
`np.exp(x)`	elementwise exponential
`np.log(x)`	elementwise natural logarithm
`np.abs(x)`	elementwise absolute value
`np.sin(x)`	elementwise sine
`np.cos(x)`	elementwise cosine
`np.tan(x)`	elementwise tangent
`np.arcsin(x)`	elementwise inverse sine
`np.arccos(x)`	elementwise inverse cosine
`np.arctan(x)`	elementwise inverse tangent


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
In [99]: A_list = [4,5,6]
A_arr = np.array(A_list)
B_list = [10,11,12]
B_arr = np.array(B_list)
print(A_list + B_list)
print(A_arr + B_arr)
[4, 5, 6, 10, 11, 12]
[14 16 18]
In [100]: A = np.array([3.2, 4.8, 1.1])
B = np.array([1.2, 10.3, 9.9])
print(A - 1)
[2.2 3.8 0.1]
Mathematical functions


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
In [99]: A_list = [4,5,6]
A_arr = np.array(A_list)
B_list = [10,11,12]
B_arr = np.array(B_list)
print(A_list + B_list)
print(A_arr + B_arr)
[4, 5, 6, 10, 11, 12]
[14 16 18]
In [100]: A = np.array([3.2, 4.8, 1.1])
B = np.array([1.2, 10.3, 9.9])
print(A - 1)
print(A *)
[2.2 3.8 0.1]

jupyter psi2018-2019_programminginpython_template_2 (autosaved)

```
B_list = [10,11,12]
B_arr = np.array(B_list)

print( A_list + B_list )
print( A_arr + B_arr )
```

```
[4, 5, 6, 10, 11, 12]
[14 16 18]
```

In [101]:
```
A = np.array([3.2, 4.8, 1.1])
B = np.array([1.2, 10.3, 9.9])

print( A - 1 )
print( A * B )
```

```
[2.2 3.8 0.1]
[ 3.84 49.44 10.8
```

Mathematical functio

Numpy has several bu cal functions that act elementwise on arrays x, such as:

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

`np.sqrt(x)`	elementwise square-root
`np.exp(x)`	elementwise exponential
`np.log(x)`	elementwise natural logarithm
`np.abs(x)`	elementwise absolute value
`np.sin(x)`	elementwise sine
`np.cos(x)`	elementwise cosine
`np.tan(x)`	elementwise tangent
`np.arcsin(x)`	elementwise inverse sine
`np.arccos(x)`	elementwise inverse cosine
`np.arctan(x)`	elementwise inverse tangent

In [30]:
```
x = np.array([0.2, 1.2, 0.56, 2.1])

print( np.sq)
```

Some other useful mathematical functions for mathematics and statistics are:

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

Logout

Numpy has several built-in mathematical functions that act elementwise on arrays x, such as:

NumPy function	Description
`np.sqrt(x)`	elementwise square-root
`np.exp(x)`	elementwise exponential
`np.log(x)`	elementwise natural logarithm
`np.abs(x)`	elementwise absolute value
`np.sin(x)`	elementwise sine
`np.cos(x)`	elementwise cosine
`np.tan(x)`	elementwise tangent
`np.arcsin(x)`	elementwise inverse sine
`np.arccos(x)`	elementwise inverse cosine
`np.arctan(x)`	elementwise inverse tangent

```
In [104]: x = np.array([0.2, 1.2, 0.56, 2.1])
```

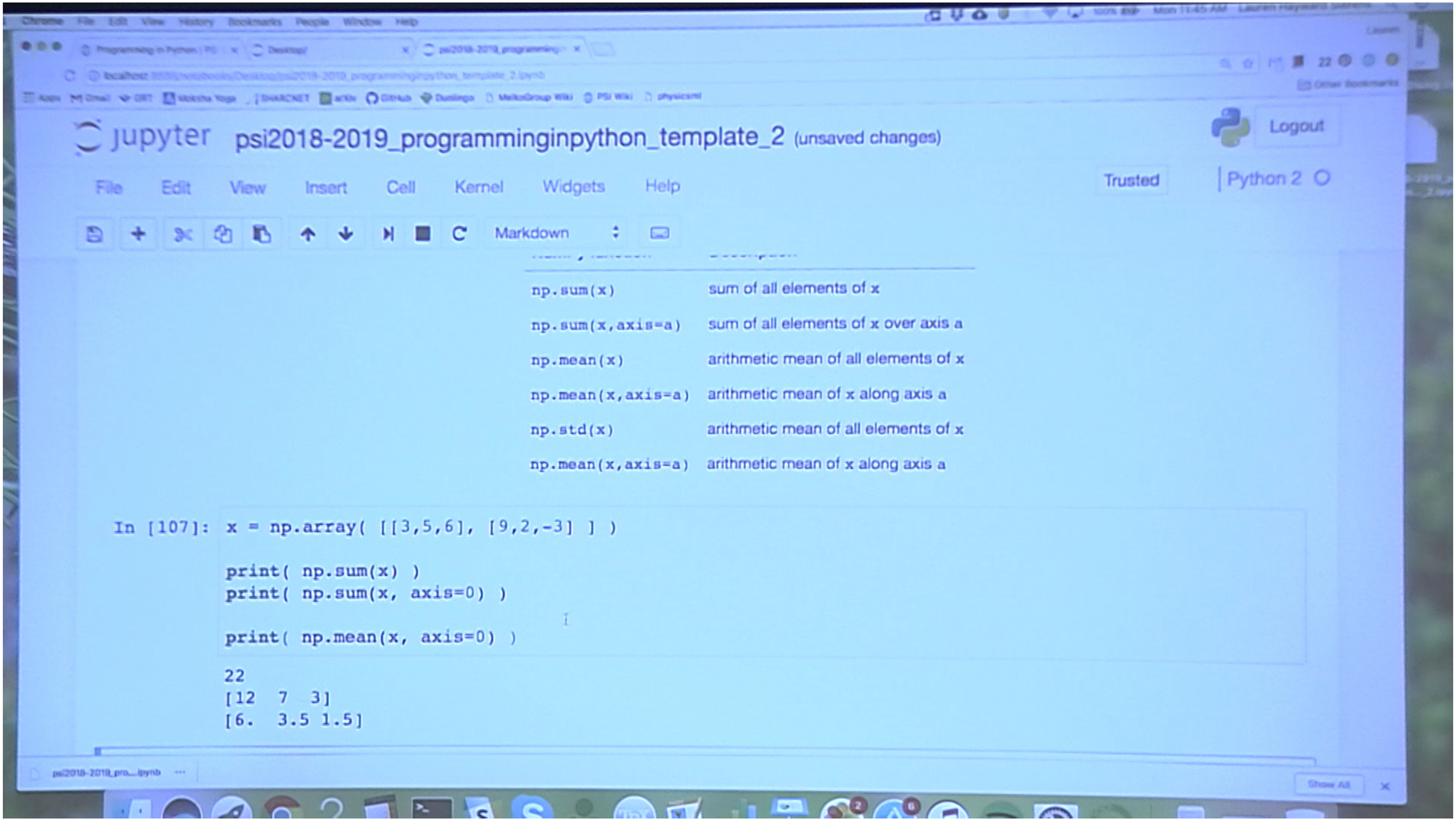
jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

NumPy function	Description
`np.sum(x)`	sum of all elements of x
`np.sum(x,axis=a)`	sum of all elements of x over axis a
`np.mean(x)`	arithmetic mean of all elements of x
`np.mean(x,axis=a)`	arithmetic mean of x along axis a
`np.std(x)`	arithmetic mean of all elements of x
`np.mean(x,axis=a)`	arithmetic mean of x along axis a

```
In [31]: x = np.array( [[3,5,6], [9,2,-3]] )

         print( np.sum(x) )
```

Exercise: Define a function that will take as input two NumPy arrays of the same size and output the elementwise sum of their squared elements.

```
In [ ]:
```


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Markdown
np.sum(x) sum of all elements of x
np.sum(x,axis=a) sum of all elements of x over axis a
np.mean(x) arithmetic mean of all elements of x
np.mean(x,axis=a) arithmetic mean of x along axis a
np.std(x) arithmetic mean of all elements of x
np.mean(x,axis=a) arithmetic mean of x along axis a
In [107]: x = np.array([[3,5,6], [9,2,-3]])
print(np.sum(x))
print(np.sum(x, axis=0))
print(np.mean(x, axis=0))
22
[12 7 3]
[6. 3.5 1.5]

psi2018-2019_programminginpython_template_2 (unsaved changes)

`np.sum(x,axis=a)`	sum of all elements of x over axis a
`np.mean(x)`	arithmetic mean of all elements of x
`np.mean(x,axis=a)`	arithmetic mean of x along axis a
`np.std(x)`	arithmetic mean of all elements of x
`np.mean(x,axis=a)`	arithmetic mean of x along axis a

In [107]:

```
x = np.array( [[3,5,6], [9,2,-3] ] )

print( np.sum(x) )
print( np.sum(x, axis=0) )

print( np.mean(x, axis=0) )
print( np.mean(x, axis=0) )
```

```
22
[12  7  3]
[6.  3.5 1.5]
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

`np.mean(x,axis=a)` arithmetic mean of x along axis a

```
In [109]: x = np.array( [[3,5,6], [9,2,-3] ] )

          print( np.sum(x) )
          print( np.sum(x, axis=0) )

          print( np.mean(x, axis=0) )

          print( np.std(x) )

22
[12  7  3]
[6.  3.5 1.5]
3.7267799624996494
```

Exercise: Define a function that will take as input two NumPy arrays of the same size and output the elementwise sum of their squared elements.

```
In [ ]:
```

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

Markdown

4.2. Random numbers

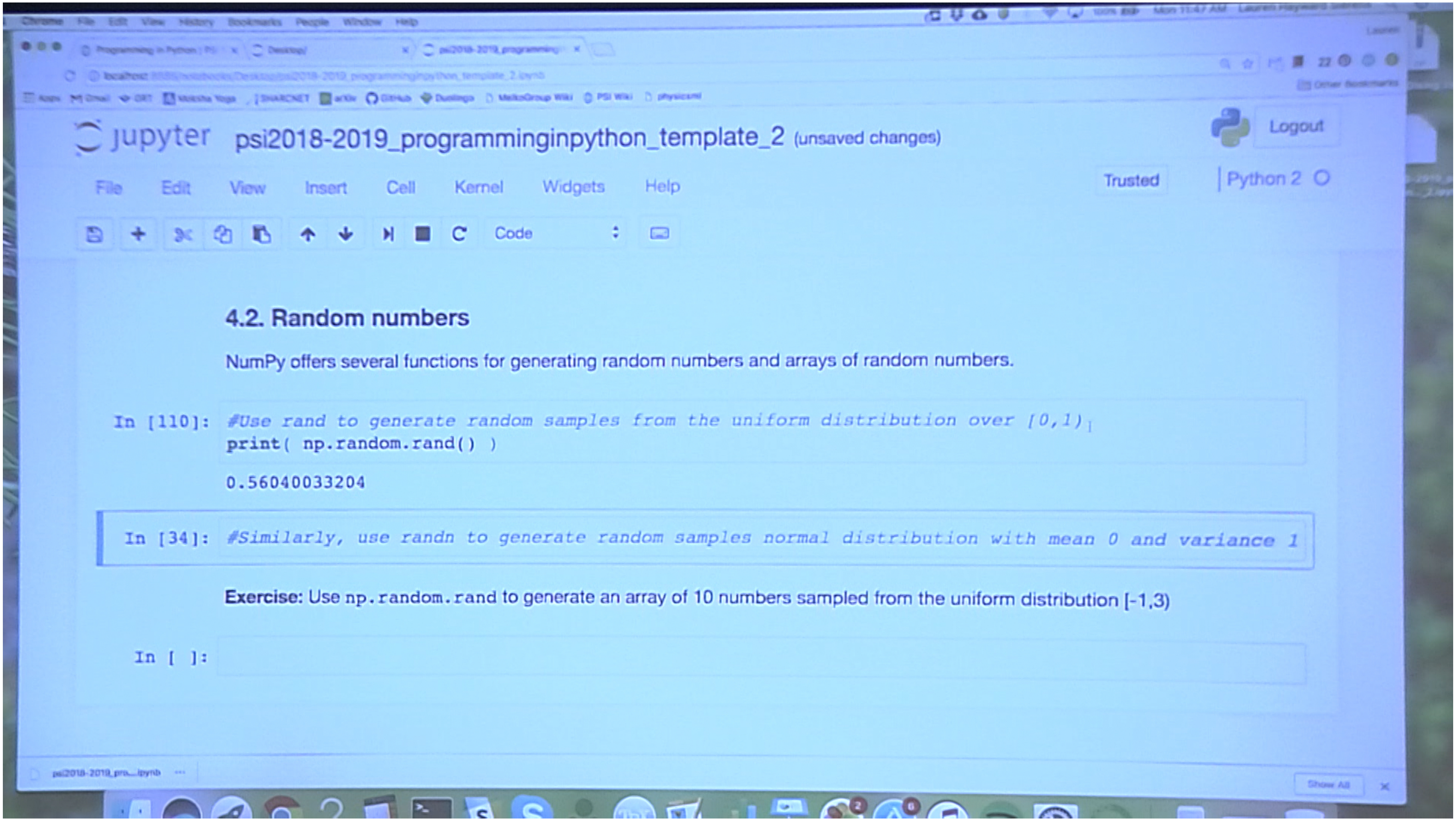
NumPy offers several functions for generating random numbers and arrays of random numbers.

In [33]:
```
#Use rand to generate random samples from the uniform distribution over [0,1)
```

In [34]:
```
#Similarly, use randn to generate random samples normal distribution with mean 0 and variance 1
```

Exercise: Use `np.random.rand` to generate an array of 10 numbers sampled from the uniform distribution [-1,3)

In [ ]:


jupyter
psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 2
Code
4.2. Random numbers
NumPy offers several functions for generating random numbers and arrays of random numbers.
In [110]:
#Use rand to generate random samples from the uniform distribution over [0,1)
print(np.random.rand())
0.56040033204
In [34]:
#Similarly, use randn to generate random samples normal distribution with mean 0 and variance 1
Exercise: Use np.random.rand to generate an array of 10 numbers sampled from the uniform distribution [-1,3)
In [ ]:

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

NumPy offers several functions for generating random numbers and arrays of random numbers.

In [113]:
```
#Use rand to generate random samples from the uniform distribution over [0,1)
print( np.random.rand() )
print( np.random.rand(5) )
print( np.random.rand(4,2) )
```

```
0.113822818921
[0.90273497 0.11473723 0.70942207 0.06108706 0.73449475]
[[0.12100888 0.70409352]
 [0.07289896 0.54196251]
 [0.92221933 0.60901518]
 [0.37251586 0.85721873]]
```

In [34]:
```
#Similarly, use randn to generate random samples normal distribution with mean 0 and variance 1
print( np.random.rand(5) )
```

Exercise: Use `np.random.rand` to generate an array of 10 numbers sampled from the uniform distribution [-1,3)

In [ ]:

jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 2

NumPy offers several functions for generating random numbers and arrays of random numbers.

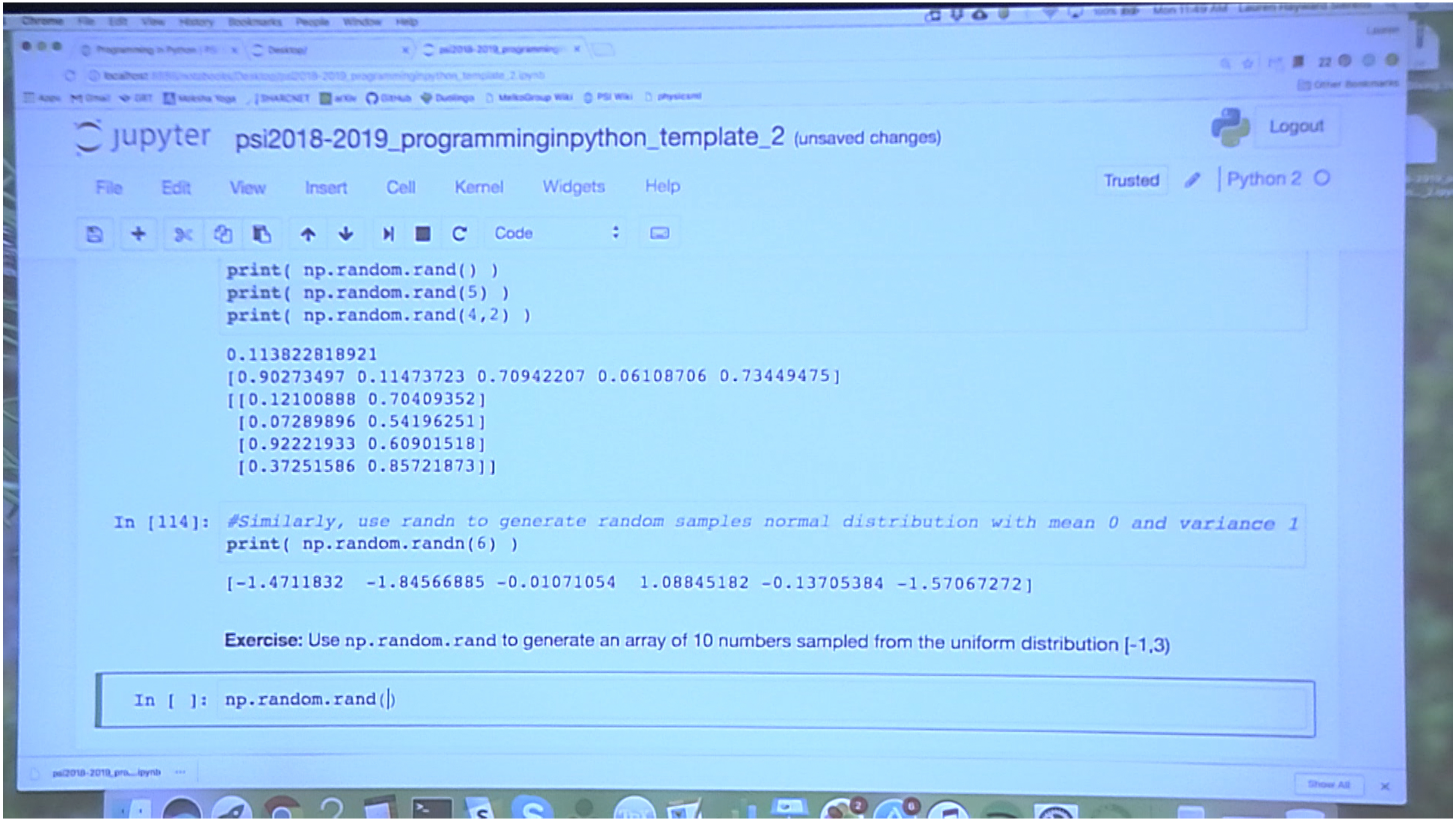
In [113]:
```python
#Use rand to generate random samples from the uniform distribution over [0,1)
print( np.random.rand() )
print( np.random.rand(5) )
print( np.random.rand(4,2) )
```

```
0.113822818921
[0.90273497 0.11473723 0.70942207 0.06108706 0.73449475]
[[0.12100888 0.70409352]
 [0.07289896 0.54196251]
 [0.92221933 0.60901518]
 [0.37251586 0.85721873]]
```

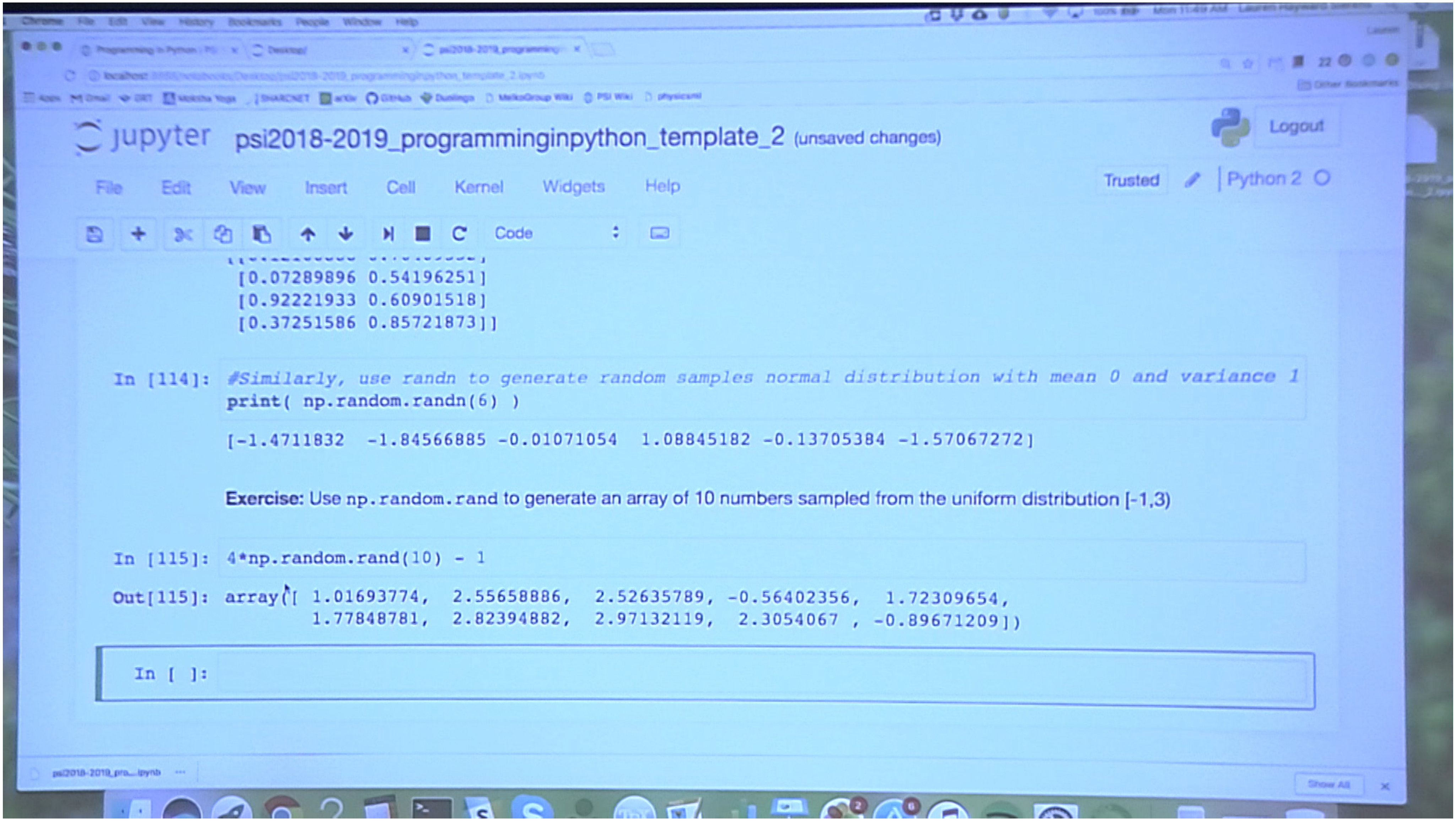
In [114]:
```python
#Similarly, use randn to generate random samples normal distribution with mean 0 and variance 1
print( np.random.randn(6) )
```

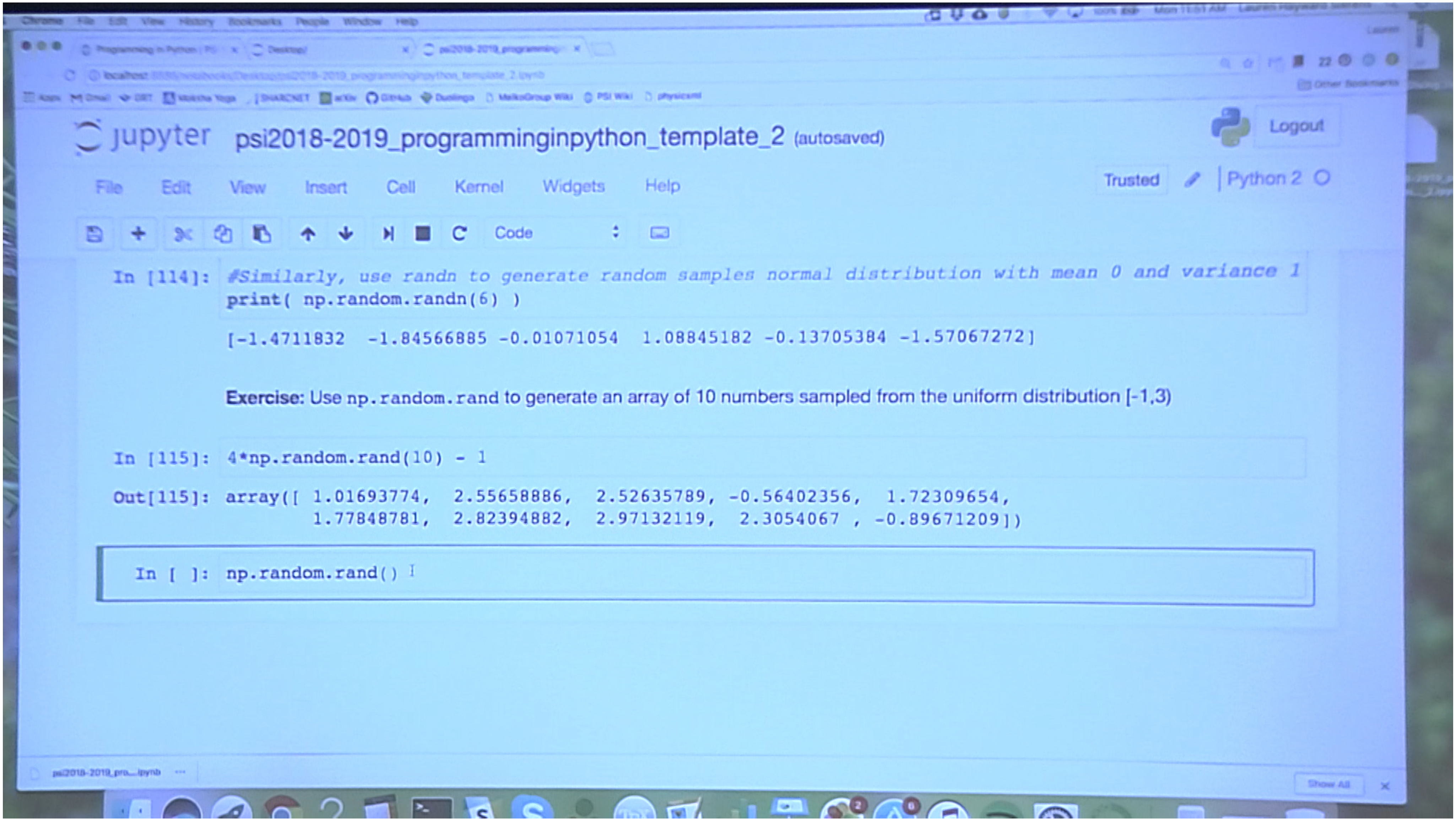
```
[-1.4711832  -1.84566885 -0.01071054  1.08845182 -0.13705384 -1.57067272]
```

Exercise: Use `np.random.rand` to generate an array of 10 numbers sampled from the uniform distribution [-1,3)



jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
print(np.random.rand())
print(np.random.rand(5))
print(np.random.rand(4,2))
0.113822818921
[0.90273497 0.11473723 0.70942207 0.06108706 0.73449475]
[[0.12100888 0.70409352]
[0.07289896 0.54196251]
[0.92221933 0.60901518]
[0.37251586 0.85721873]]
In [114]: #Similarly, use randn to generate random samples normal distribution with mean 0 and variance 1
print(np.random.randn(6))
[-1.4711832 -1.84566885 -0.01071054 1.08845182 -0.13705384 -1.57067272]
Exercise: Use np.random.rand to generate an array of 10 numbers sampled from the uniform distribution [-1,3)
In [ ]: np.random.rand()


jupyter psi2018-2019_programminginpython_template_2 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
[0.07289896 0.54196251]
[0.92221933 0.60901518]
[0.37251586 0.85721873]]
In [114]: #Similarly, use randn to generate random samples normal distribution with mean 0 and variance 1
print(np.random.randn(6))
[-1.4711832 -1.84566885 -0.01071054 1.08845182 -0.13705384 -1.57067272]
Exercise: Use np.random.rand to generate an array of 10 numbers sampled from the uniform distribution [-1,3)
In [115]: 4*np.random.rand(10) - 1
Out[115]: array([1.01693774, 2.55658886, 2.52635789, -0.56402356, 1.72309654,
1.77848781, 2.82394882, 2.97132119, 2.3054067 , -0.89671209])
In [ ]:


jupyter psi2018-2019_programminginpython_template_2 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 2
Code
In [114]: #Similarly, use randn to generate random samples normal distribution with mean 0 and variance 1
print(np.random.randn(6))
[-1.4711832 -1.84566885 -0.01071054 1.08845182 -0.13705384 -1.57067272]
Exercise: Use np.random.rand to generate an array of 10 numbers sampled from the uniform distribution [-1,3)
In [115]: 4*np.random.rand(10) - 1
Out[115]: array([1.01693774, 2.55658886, 2.52635789, -0.56402356, 1.72309654,
1.77848781, 2.82394882, 2.97132119, 2.3054067 , -0.89671209])
In [ ]: np.random.rand()